UniSENAI

Módulo 7: Aplicações Avançadas com Node.js

Módulo 7: Aplicações Avançadas com Node.js

Objetivo:

Explorar funcionalidades avançadas do Node.js para capacitar os alunos a desenvolver aplicações modernas e escaláveis com alta performance.

Conteúdo Detalhado:

Aula 7.1: Programação assíncrona com Promises e async/await

O que é Programação Assíncrona?

- Permite que o código execute tarefas sem bloquear o fluxo principal.
- Torna as aplicações mais eficientes ao lidar com I/O.

Promises

- Representam a eventual conclusão (ou falha) de uma operação assíncrona.
- Exemplo:

```
const promessa = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Sucesso!'), 1000);
});

promessa.then(console.log).catch(console.error);
```

Async/Await

- Sintaxe mais simples e legível para lidar com promises.
- Exemplo:

```
async function executar() {
  try {
    const resultado = await promessa;
    console.log(resultado);
  } catch (erro) {
    console.error(erro);
  }
}

executar();
```

Aula 7.2: Trabalhando com worker_threads para multitarefa

O que é Worker Threads?

- Permite executar código JavaScript em threads separados.
- Ideal para tarefas pesadas que não devem bloquear o Event Loop.

Exemplo Simples

1. **Arquivo principal (`index.js`)**

```
const { Worker } = require('worker_threads');

const worker = new Worker('./worker.js');
worker.on('message', (mensagem) => console.log('Mensagem do Worker:',
mensagem));
```

2. **Arquivo do Worker (`worker.js`)**

```
const { parentPort } = require('worker_threads');

parentPort.postMessage('Tarefa concluída no Worker!');
```

Aula 7.3: Integração de Websockets com Socket.io

O que é Socket.io?

- Uma biblioteca que facilita a comunicação bidirecional em tempo real entre cliente e servidor.

Instalação

```
npm install socket.io
```

Exemplo Básico

1. **Servidor (`server.js`)**

```
const express = require('express');
const http = require('http');
const { Server } = require('socket.io');

const app = express();
const server = http.createServer(app);
const io = new Server(server);

io.on('connection', (socket) => {
  console.log('Usuário conectado:', socket.id);
  socket.on('mensagem', (msg) => console.log('Mensagem recebida:', msg));
});

server.listen(3000, () => console.log('Servidor rodando na porta 3000'));
```

2. **Cliente (HTML)**

```
<script src="/socket.io/socket.io.js"></script>
<script>
  const socket = io();
  socket.emit('mensagem', 'Olá do cliente!');
</script>
```

Aula 7.4: Estratégias de escalabilidade com clusters e child processes

Clusters

- Permitem aproveitar todos os núcleos do processador para melhorar a performance.

Exemplo de Cluster

```
const cluster = require('cluster');
const http = require('http');
const numCPUs = require('os').cpus().length;

if (cluster.isMaster) {
  console.log(`Processo mestre: ${process.pid}`);
  for (let i = 0; i < numCPUs; i++) {
    cluster.fork();
  }
} else {
  http.createServer((req, res) => res.end('Cluster ativo')).listen(8000);
}
```

Child Processes

- Permite criar processos filhos para tarefas independentes.

Exemplo de Child Process

```
const { spawn } = require('child_process');

const processo = spawn('ls', ['-lh']);
processo.stdout.on('data', (data) => console.log(`Saída: ${data}`));
```

Aula 7.5: Monitoramento e logging com PM2 e Winston

PM2

- Gerenciador de processos que facilita o monitoramento e a manutenção de aplicações Node.js.
- Instalação:

```
npm install pm2 -g
```

- Iniciar um processo:

```
pm2 start app.js
```

Winston

- Biblioteca para registro de logs estruturados.

Exemplo com Winston

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

const logger = winston.createLogger({
  level: 'info',
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'app.log' })
  ]
});

logger.info('Informação registrada no log!');
```

Lista de Exercícios

Questões Teóricas

1. Explique a diferença entre callbacks, promises e async/await.
2. O que é o Worker Threads no Node.js e qual sua utilidade?
3. Como o Socket.io facilita a comunicação em tempo real?
4. Qual é o papel dos clusters em aplicações Node.js?
5. Liste três vantagens do PM2 para o gerenciamento de processos.
6. O que é um child process e como ele é utilizado?
7. Explique o conceito de logging estruturado e sua importância.
8. Qual é a vantagem de usar o Winston para logs?
9. Por que é importante escalar aplicações Node.js em ambientes de produção?
10. Como a programação assíncrona melhora a performance de aplicações?

Questões Práticas

1. Implemente uma função que utilize async/await para buscar dados de uma API.
2. Configure um Worker Thread que execute uma tarefa independente.
3. Crie um servidor Socket.io que receba mensagens de um cliente.
4. Implemente um cluster que utilize todos os núcleos do processador.
5. Configure um processo filho que execute um comando do sistema operacional.
6. Instale o PM2 e monitore uma aplicação Node.js.
7. Configure um logger com Winston para salvar logs em arquivo.
8. Crie uma aplicação que registre logs de erros e informações.
9. Use o Socket.io para criar um chat em tempo real.
10. Combine Worker Threads e logging para monitorar tarefas em segundo plano.

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).