UniSENAI

Módulo 5: Server-Side Rendering SSR, Templating e Interface com Servidor

Módulo 5: Server-Side Rendering, Templating e Interface com Servidor

---

Objetivo:

Capacitar os alunos a renderizar páginas dinâmicas utilizando dados provenientes do MySQL, com foco em organização de layouts e integração eficiente entre front-end e back-end.

Conteúdo Detalhado:

5.1 SSR (Server-Side Rendering) no Node.js

O que é SSR (Server-Side Rendering)?

Server-Side Rendering (SSR) é uma técnica de renderização de páginas web em que o código HTML é gerado no servidor e enviado ao cliente (navegador). Em contraste com o Client-Side Rendering (CSR), onde a renderização é feita no navegador, o SSR oferece vantagens como:

- **Melhoria no SEO**: Motores de busca conseguem indexar o conteúdo imediatamente, pois ele já está renderizado.
- **Carregamento mais rápido para o usuário**: O HTML inicial chega ao navegador já processado.
- **Melhor desempenho em dispositivos de baixa potência**: Reduz a carga de processamento no cliente.

Como funciona o SSR?

1. **Requisição do cliente**: O navegador do usuário faz uma requisição HTTP ao servidor.
2. **Renderização no servidor**: O servidor processa a requisição, gera o HTML dinâmico e retorna ao cliente.
3. **Entrega ao cliente**: O navegador recebe o HTML completo e o exibe imediatamente.
4. **Hidratação (opcional)**: Caso o SSR seja combinado com frameworks de frontend (como React), o JavaScript carrega e torna a página interativa.

Tecnologias e Ferramentas Usadas com Node.js para SSR

1. Express.js

Express é uma biblioteca minimalista que simplifica a criação de servidores Node.js.

2. Templates Engines

Engines de template geram HTML dinâmico baseado em dados e lógica de renderização. Exemplos populares incluem:

- **Pug (anteriormente Jade)**
- **Handlebars**
- **EJS (Embedded JavaScript)**

3. SSR com Frameworks de Frontend

Frameworks modernos como React, Vue.js e Next.js possuem suporte para SSR.

Exemplo Prático: Implementação de SSR com Node.js e Express

1. Instalando Dependências

```
npm init -y
npm install express ejs
```

2. Estrutura do Projeto

```
project/
  |-- views/
      |-- index.ejs
  |-- server.js
```

3. Criando o Servidor

Arquivo: `server.js`

```
const express = require('express');
const app = express();
const PORT = 3000;

// Configurar a engine de template EJS
app.set('view engine', 'ejs');

// Definir rota principal
app.get('/', (req, res) => {
    const data = { title: 'SSR com Node.js', message: 'Bem-vindo ao SSR!' };
    res.render('index', data);
});

// Iniciar servidor
app.listen(PORT, () => {
    console.log(`Servidor rodando em http://localhost:${PORT}`);
});
```

4. Criando o Template

Arquivo: `views/index.ejs`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= title %></title>
</head>
<body>
    <h1><%= message %></h1>
</body>
</html>
```

5. Executando o Projeto

```
node server.js
```

Abra o navegador em `http://localhost:3000` e veja a página renderizada pelo servidor.

Aplicabilidade do SSR

- **Sites orientados a conteúdo**: Blogs, portais de notícias e sites de e-commerce.
- **SEO é crucial**: Sites que dependem de visibilidade em motores de busca.
- **Dispositivos com hardware limitado**: SSR reduz a carga de renderização no cliente.
- **Páginas que necessitam de carregamento rápido inicial**.

Atividades Propostas

Teóricas:

1. Explique as diferenças entre Server-Side Rendering (SSR) e Client-Side Rendering (CSR).
2. Liste três vantagens e três desvantagens do SSR.
3. Qual é o impacto do SSR no SEO? Justifique com exemplos.
4. Explique a diferença entre hidratação e renderização inicial no SSR.
5. Pesquise e explique como o Next.js utiliza SSR para melhorar a experiência do desenvolvedor.

Práticas:

6. Modifique o exemplo prático para incluir uma lista de produtos no template `index.ejs`. A lista deve ser gerada dinamicamente no servidor.
7. Adicione uma nova rota `/about` que também use SSR para renderizar um template com informações fictícias sobre o site.
8. Utilize a biblioteca Handlebars como template engine em vez de EJS.
9. Integre uma API externa (ex.: JSONPlaceholder) e exiba os dados obtidos no template.
10. Combine SSR com CSR utilizando React para criar uma página interativa. O SSR deve fornecer o HTML inicial, e o React deve tornar a página dinâmica após a hidratação.

5.2: Introdução às Templating Engines (EJS, Pug)

O que são Templating Engines?

- Ferramentas que permitem criar páginas HTML dinâmicas no lado do servidor.
- Permitem a injeção de dados em tempo real e organização de layouts reutilizáveis.

EJS (Embedded JavaScript):

- Sintaxe similar ao HTML.
- Tags de uso: `<%= %>` para saída de variáveis e `<% %>` para código JavaScript.

Pug:

- Sintaxe minimalista baseada em indentção.
- Focado em simplicidade e organização.

Exemplo de Comparativo:

- **EJS:**

```
<h1> Olá, <%= nome %>! </h1>
```

- **Pug:**

```
h1 Olá, #{nome}!
```

5.3: Configuração de templates dinâmicos com EJS

Instalação do EJS:

```
npm install ejs
```

Configuração no Express:

```
const express = require('express');
const app = express();

app.set('view engine', 'ejs');
app.set('views', './views');
```

Criando um Template Simples:

1. Crie a pasta `views`.
2. Adicione o arquivo `index.ejs`:

```
<!DOCTYPE html>
<html>
<head>
  <title>Página Dinâmica</title>
</head>
<body>
  <h1> Olá, <%= nome %>! </h1>
</body>
</html>
```

Renderizando o Template:

```
app.get('/', (req, res) => {
  res.render('index', { nome: 'João' });
});
```

5.4: Renderizando dados do MySQL em páginas dinâmicas

Conexão ao Banco e Envio de Dados:

```
const mysql = require('mysql2');
const connection = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'sua_senha',
  database: 'projeto_node'
});

app.get('/usuarios', (req, res) => {
  connection.query('SELECT * FROM usuarios', (err, results) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.render('usuarios', { usuarios: results });
  });
});
```

Template Exemplo (`usuarios.ejs`):

```
<!DOCTYPE html>
<html>
<head>
  <title>Lista de Usuários</title>
</head>
<body>
  <h1>Usuários</h1>
  <ul>
    <% usuarios.forEach(usuario => { %>
      <li><%= usuario.nome %> - <%= usuario.email %></li>
    <% }) %>
  </ul>
</body>
</html>
```

5.5: Trabalhando com partials e layouts reutilizáveis

O que são Partials?

- Arquivos reutilizáveis que contêm partes de um layout comum, como cabeçalhos ou rodapés.

Criando um Partial (`header.ejs`):

```
<header>
  <h1>Bem-vindo ao Sistema</h1>
</header>
```

Incluindo Partials no Template:

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemplo com Partials</title>
</head>
<body>
  <% include ./partials/header %>
  <main>
    <h2>Conteúdo Principal</h2>
  </main>
</body>
</html>
```

Layout Reutilizável com Placeholder:

```
<!DOCTYPE html>
<html>
<head>
  <title><%= title %></title>
</head>
<body>
  <% include ./partials/header %>
  <%= content %>
</body>
</html>
```

5.6: Formulários no Express: Validação e envio de dados para o MySQL

Configuração para Processar Formulários:

- Instale o middleware `body-parser`:

```
npm install body-parser
```

- Configure no Express:

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));
```

Criação de Formulário:

```
<form action="/usuarios" method="POST">
  <input type="text" name="nome" placeholder="Nome" required>
  <input type="email" name="email" placeholder="Email" required>
  <button type="submit">Enviar</button>
</form>
```

Rota para Receber Dados:

```
app.post('/usuarios', (req, res) => {
  const { nome, email } = req.body;
  connection.query('INSERT INTO usuarios (nome, email) VALUES (?, ?)', [nome, email],
(err) => {
    if (err) {
      res.status(500).send(err);
      return;
    }
    res.redirect('/usuarios');
  });
});
```

Lista de Exercícios

Questões Teóricas

1. O que são templating engines e qual sua utilidade?
2. Explique as diferenças entre EJS e Pug.
3. Como configurar o EJS em um projeto Express?
4. O que são partials e como eles auxiliam na reutilização de código?
5. Como os dados do MySQL podem ser renderizados em um template?
6. Explique o papel do `body-parser` em aplicações Express.
7. Quais são as boas práticas ao organizar templates em um projeto?
8. Como funciona a inclusão de placeholders em layouts do EJS?
9. Liste três aplicações práticas para o uso de formulários no Express.
10. Por que é importante validar dados recebidos de formulários?

Questões Práticas

1. Configure o EJS em um projeto Express e crie um template básico.
2. Implemente um template que liste usuários cadastrados em um banco MySQL.
3. Crie um layout com cabeçalho e rodapé utilizando partials.
4. Desenvolva um formulário para cadastrar novos usuários no banco de dados.
5. Configure uma rota POST que receba os dados do formulário e insira no MySQL.
6. Crie um template que exiba mensagens de erro ao processar dados.
7. Implemente um sistema de navegação entre páginas utilizando layouts reutilizáveis.
8. Crie um placeholder dinâmico para o título da página em um layout.
9. Integre dados do MySQL em um menu dinâmico renderizado via EJS.
10. Valide os campos de entrada do formulário e exiba mensagens de erro caso os dados sejam inválidos.

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).
   -