Roteiro de Atividade Prática

Nome: Turma: .

**Atividade: Otimização de performance em aplicações *front-end***

Você foi contratado por uma empresa de desenvolvimento de software para otimizar a performance de uma aplicação web em React. A aplicação apresenta problemas de lentidão ao carregar páginas com muitos componentes e imagens grandes. O objetivo desta atividade é implementar três técnicas de otimização: **Lazy Loading**, **Code Splitting** e **Minificação**. Após a implementação, você deverá medir o impacto dessas técnicas na performance da aplicação.

Situação fictícia produzida pela SEDUC-SP.

**1. Preparação do ambiente**

Antes de iniciar a otimização, certifique-se de que a aplicação React já está configurada. Se ainda não tem uma aplicação, crie uma com **Create React App**:

npx create-react-app performance-optimization

cd performance-optimization

**2. Lazy Loading de imagens e componentes**

**Objetivo**: Carregar imagens e

componentes apenas quando necessário, melhorando o tempo de carregamento inicial da página.

**2.1 Lazy loading de imagens**

* **Passo a passo**:
  1. Localize todas as tags <img> no seu código.
  2. Adicione o atributo loading="lazy" a essas imagens:

html

<img src="path-to-image.jpg" alt="description" loading="lazy" />

* 1. Esse atributo nativo faz com que as imagens sejam carregadas apenas quando o usuário rola a página até a área onde elas estão.

Caso não tenha nada, vou propor um código exemplo (apenas para você praticar e entender ):

Edite o arquivo src/App.js para incluir o código abaixo:

import React from 'react';

function App() {

return (

<div className="App">

<h1>Lazy Loading Example</h1>

<p>Role a página para ver o carregamento da imagem.</p>

{/\* Espaço vazio para forçar rolagem \*/}

<div style={{ height: '1000px', backgroundColor: '#f0f0f0' }}>

<p>Área inicial sem imagem (Role para baixo)</p>

</div>

{/\* Imagem com Lazy Loading \*/}

<div>

<h2>Imagem com Lazy Loading:</h2>

<img

src="https://via.placeholder.com/800x400"

alt="Placeholder"

loading="lazy"

width="800"

height="400"

/>

</div>

{/\* Conteúdo adicional após a imagem \*/}

<div style={{ height: '500px', backgroundColor: '#f0f0f0' }}>

<p>Conteúdo adicional após a imagem</p>

</div>

</div>

);

}

export default App;

**Explicação desse código**

* **Lazy Loading**: o atributo loading="lazy" no elemento <img> faz com que a imagem seja carregada apenas quando o usuário rola até ela.
* **Placeholder Image**: a imagem usada é um placeholder (<https://via.placeholder.com/800x400>. Acesso em: 9 dez. 2024.), mas você pode substituí-la por uma imagem real.

4. Teste

* Quando você rodar a aplicação (npm start), verá uma página com uma área inicial sem imagens;
* Ao rolar para baixo, a imagem será carregada de forma dinâmica, otimizando o tempo de carregamento inicial.

**2.2 Lazy Loading de componentes**

* **Passo a passo**:
  1. Se já tens um site, Identifique componentes que podem ser carregados dinamicamente, como seções de uma página ou módulos secundários.
  2. Utilize React.lazy para dividir o carregamento desses componentes. Por exemplo:

const LazyComponent = React.lazy(() => import('./LazyComponent'));

function App() {

return (

<div>

<h1>Bem-vindo à Aplicação</h1>

<Suspense fallback={<div>Carregando...</div>}>

<LazyComponent />

</Suspense>

</div>

);

}

* 1. O componente só será carregado quando o usuário acessar essa parte da aplicação, o que diminui o tamanho inicial do arquivo JavaScript.

Um exemplo prático de Lazzy com componentes:

**Instale o pacote react-router-dom:** no terminal, dentro do diretório do seu projeto, execute o seguinte comando para instalar o react-router-dom:

npm install react-router-dom

Esse exemplo de **React Lazy com suspense** seria colocado no arquivo principal da sua aplicação, geralmente chamado de App.js, em que você normalmente define a estrutura de roteamento e componentes principais.

Aqui está como organizar o código em um projeto típico de React criado com **Create React App**:

**Estrutura do projeto:**

performance-optimization/

│

├── src/

│ ├── components/

│ │ ├── Home.js

│ │ └── About.js

│ ├── App.js

│ ├── index.js

│ └── ... (outros arquivos)

└── public/

└── ... (outros arquivos)

**Passo a passo:**

1. **Crie os componentes Home e About:** No diretório src/components, crie os arquivos Home.js e About.js.

**Home.js**:

import React from 'react';

function Home() {

return (

<div>

<h2>Página Inicial</h2>

<p>Bem-vindo à Home Page!</p>

</div>

);

}

export default Home;

**About.js**:

import React from 'react';

function About() {

return (

<div>

<h2>Sobre Nós</h2>

<p>Essa é a página Sobre Nós da aplicação.</p>

</div>

);

}

export default About;

1. **Configure o React Lazy no App.js:**

No arquivo src/App.js, é onde você vai aplicar o React Lazy e Suspense para carregar os componentes de forma dinâmica.

**App.js**:

import React, { Suspense } from 'react';

import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';

// Lazy loading dos componentes

const Home = React.lazy(() => import('./components/Home'));

const About = React.lazy(() => import('./components/About'));

function App() {

return (

<Router>

<Suspense fallback={<div>Carregando...</div>}>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</Suspense>

</Router>

);

}

export default App;

1. **Configuração de roteamento (index.js):** No arquivo src/index.js, você não precisa fazer nenhuma modificação específica relacionada ao lazy loading. A configuração básica do React já está pronta para lidar com roteamento e renderização.

**index.js**:

import React from 'react';

import ReactDOM from 'react-dom';

import './index.css';

import App from './App';

ReactDOM.render(

<React.StrictMode>

<App />

</React.StrictMode>,

document.getElementById('root')

);

1. **Executar a aplicação:** Agora, execute o projeto usando o comando:

npm start

* + Ao iniciar a aplicação, a página **Home** será carregada;
  + Quando você navegar até a página **About** (/about), o componente **About** será carregado dinamicamente usando **React.lazy**;
  + Você pode observar o comportamento de carregamento assíncrono no painel de rede (Network) das **DevTools** do navegador.

**3. Minificação**

**Objetivo**: Reduzir o tamanho dos arquivos JavaScript, CSS e HTML removendo espaços em branco, comentários e outros caracteres desnecessários.

**3.1 Minificando JavaScript com Terser**

* **Passo a passo**:
  1. Se estiver utilizando Webpack, adicione a minificação no arquivo webpack.config.js:

const TerserPlugin = require('terser-webpack-plugin');

module.exports = {

mode: 'production',

optimization: {

minimize: true,

minimizer: [new TerserPlugin()],

},

};

* 1. Isso garante que seu código JavaScript será minificado automaticamente quando você fizer a build da aplicação.

**3.2 Minificando CSS com CSSNano**

* **Passo a passo**:
  1. Adicione o plugin CSSNano ao Webpack para minificar seus arquivos CSS; para isso, digite no prompt:

npm install cssnano --save-dev

* 1. Inclua o seguinte no arquivo webpack.config.js:

const CssMinimizerPlugin = require('css-minimizer-webpack-plugin');

module.exports = {

optimization: {

minimize: true,

minimizer: [

`...`,

new CssMinimizerPlugin(),

],

},

};

**3.3 Minificação no Create React App**

Se você estiver usando **Create React App**, a minificação é automaticamente feita ao rodar (prompt):

npm run build

Essa build já inclui a minificação do JavaScript, CSS e HTML.

**4. Medição de performance**

**Objetivo**: Comparar o desempenho antes e depois das otimizações.

**4.1 Utilizando Google Lighthouse**

* **Passo a passo**:
  1. Abra sua aplicação no **Google Chrome**.
  2. Vá até o DevTools (F12), selecione a aba **Lighthouse**.
  3. Clique em **Generate Report** e aguarde. Esse relatório mostrará métricas como tempo de carregamento, uso de JavaScript e CSS.
  4. Faça isso antes e depois das otimizações e compare os resultados.

**4.2 Utilizando Chrome DevTools**

* **Passo a passo**:
  1. Abra o Chrome DevTools e vá até a aba **Network**.
  2. Acesse a página da sua aplicação e observe o tamanho dos arquivos carregados, tempo de resposta e o número de requests.
  3. Use isso para monitorar a eficiência de cada técnica aplicada.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |