UNIVERSIDADE FEDERAL DO CEARÁ - CAMPUS SOBRAL

CURSO DE ENGENHARIA DE COMPUTAÇÃO

DISCIPLINA: TECNOLOGIAS WEB II

PROFESSOR: THIAGO IACHILEY ARAUJO DE SOUZA

LABORATÓRIO 02: SERVIDOR WEB SIMPLES COM MÓDULO HTTP DO NODE.JS

ALUNO: IZAIAS MACHADO PESSOA NETO

MATRÍCULA: 497372

SOBRAL

2024

SUMÁRIO

1 INTRODUÇÃO

Os objetivos deste projeto são de implementar um site básico utilizando Node.js e HTTP, que ofereça uma página inicial com produtos em destaque, detalhes dos produtos e um menu de navegação. Com isso, busca-se uma experiência prática na construção de uma aplicação web funcional. Como especificação da página foram fornecidos os *wireframes* das Figuras 1 e 2.

Figura 1 – Wireframe - Parte 01

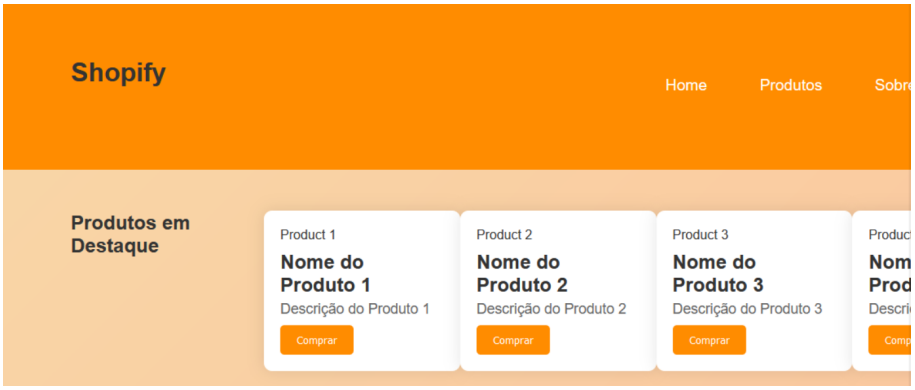
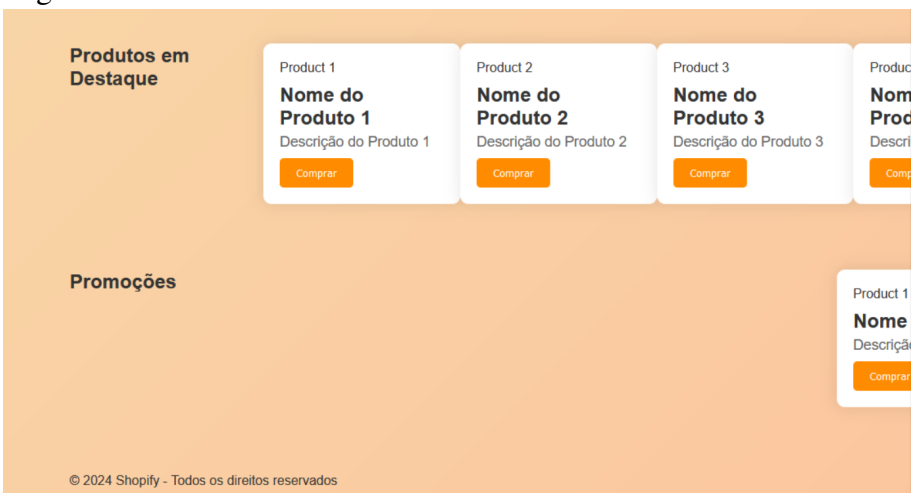


Figura 2 – Wireframe - Parte 02



Neste trabalho, é apresentado o processo detalhado do desenvolvimento deste projeto, abrangendo desde a estruturação dos arquivos, implementação das funcionalidades específicas e também instruções de uso. A fim de proporcionar uma organização clara e acessível, este trabalho foi divido em detalhes de implementação do projeto (Capítulo 2) e instruções de uso e execução do projeto (Capítulo 3).

2 IMPLEMENTAÇÃO

Este capítulo é utilizado para documentar o processo de desenvolvimento desse projeto. Nesse sentido, o capítulo foi dividido em organização dos arquivos (Seção 2.1), desenvolvimento do servidor web (Seção 2.2) e desenvolvimento da interface de usuário (Seção 2.3).

2.1 Organização dos arquivos

A Figura 3 mostra a organização dos arquivos do projeto. Note que os arquivos da pasta `public` são arquivos estáticos que o usuário pode solicitar. Além disso, a pasta `src` armazena os arquivos do lado do servidor da aplicação. Já os arquivos `package-lock` e `package` são do gerenciador de pacotes NPM.

Figura 3 – Árvore de arquivos

2.2 Desenvolvimento do código do servidor web

Na Figura 4 é feito a importação dos módulos `fs`, `path` e `http` que são utilizados durante o código. O módulo `fs` é relativo ao sistema de arquivos, em que a abreviatura `fs` vem do inglês *file system*. Já o `path` é utilizado para concatenar *strings* dos caminhos dos arquivos. E o módulo `http` é utilizado para criar servidores web simples, com Node.js.

Ainda na Figura 4, é declarada uma função que tenta buscar o conteúdo de um arquivo que está na pasta pública, utilizada para colocar todos os arquivos que o usuário tem permissão de acesso. Nesse sentido, essa função recebe o nome do arquivo e caso o arquivo exista, seu conteúdo é passado como retorno da função. Mas caso o arquivo não exista, é retornado um valor nulo.

A Figura 5 retorna dado um caminho de arquivo ou nome de arquivo o seu *mime type*, que é utilizado no envio de um arquivo na resposta de uma requisição HTTP. Caso o *mime*

Figura 4 – Código do servidor web - Parte 01

```
1   const fs = require("fs");
2   const path = require("path");
3   const http = require("http");
4   const port = process.env.PORT || 4000;
5
6   const getPublicFileContent = (file) => {
7     const publicFolder = path.join(__dirname, "..", "public");
8     const filePath = path.join(publicFolder, file);
9
10    if (!fs.existsSync(filePath)) {
11      return null;
12    }
13
14    const fileContent = fs.readFileSync(filePath, "utf-8");
15    return fileContent;
16  };
```

type do arquivo não tenha sido mapeado, o *mime type* retornado é de um texto simples.

Figura 5 – Código do servidor web - Parte 02

```
18  const getMimeType = (filename) => {
19    const fileExtension = filename.split(".")[1];
20    const mimeType = {
21      html: "text/html",
22      css: "text/css",
23      js: "text/javascript",
24    };
25
26    if (mimeType[fileExtension]) {
27      return mimeType[fileExtension];
28    }
29
30    return "text/plain";
31  };
```

Dado o caminho de um arquivo requisitado e a própria requisição, a função apresentada na Figura 6 chama funções previamente apresentadas para buscar o *mime type* e também o conteúdo do arquivo. Se não for possível encontrar o arquivo, é retornado um erro 404. Já se o arquivo for encontrado, ele é enviado ao cliente.

A instância do servidor web é declarada na Figura 7. O servidor é configurado para que ao receber requisições com a *url* padrão /, retorne o arquivo *index.html*. Caso contrário, é tentado enviar ao usuário um arquivo com nome igual ao da *string* passada na *url*. Se não existir arquivo correspondente, é retornado um erro 404, conforme foi explicado previamente.

Figura 6 – Código do servidor web - Parte 03

```
33  const writeFileResponse = (res, file) => {
34    const mimeType = getMimeType(file);
35    const fileContent = getPublicFileContent(file);
36
37    if (!fileContent) {
38      res.writeHead(404, { "Content-Type": "text/html" });
39      return res.end("404 Not Found");
40    }
41
42    res.writeHead(200, { "Content-Type": mimeType });
43    return res.end(fileContent);
44  };
```

Figura 7 – Código do servidor web - Parte 04

```
46  const server = http.createServer((req, res) => {
47    if (req.url === "/") {
48      return writeFileResponse(res, "index.html");
49    }
50
51    const possibleFileNames = req.url.split("/")[1];
52    return writeFileResponse(res, possibleFileNames);
53  });
54
55  server.listen(port, () => {
56    console.log(`Server is listening on port ${port}`);
57  });
```

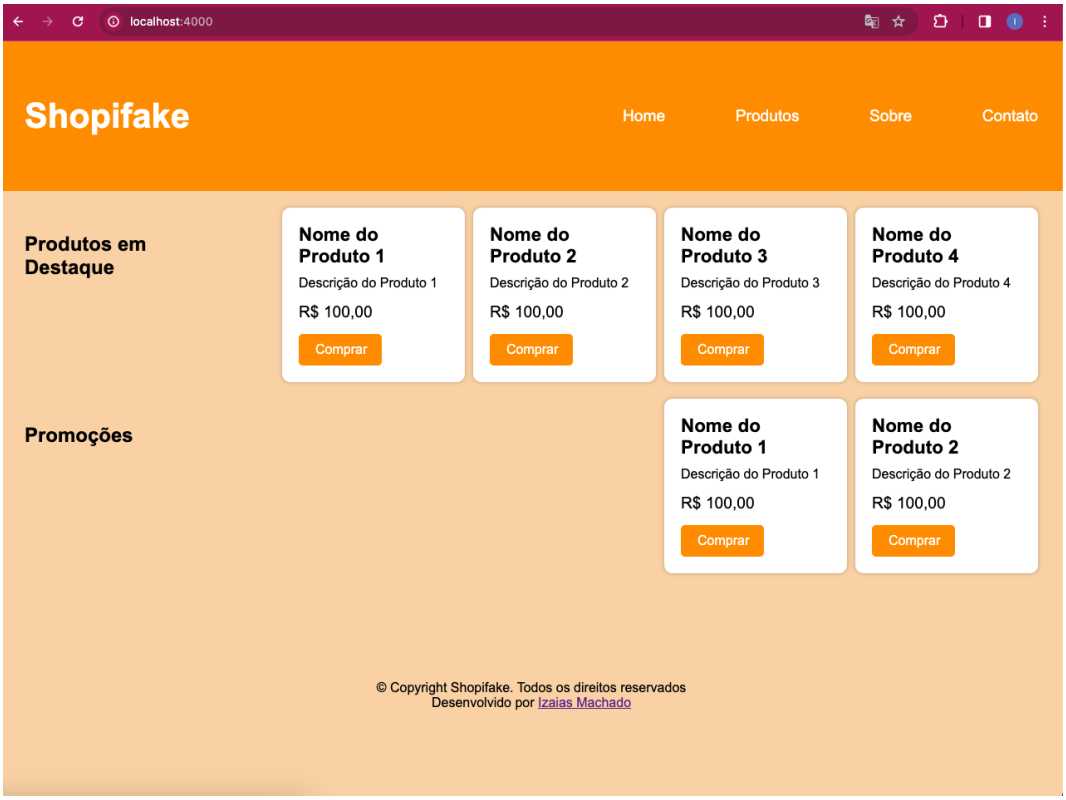
2.3 Desenvolvimento da interface de usuário

No desenvolvimento da interface de usuário, inicialmente foi criado somente o arquivo *index.html* e feita uma estrutura inicial do HTML. A Figura 8 mostra a interface de usuário nesse estágio.

Figura 8 – Página sem HTML

Já na Figura 9 é apresentada uma imagem com a página estilizada, após a criação da folha de estilos do arquivo *style.css*.

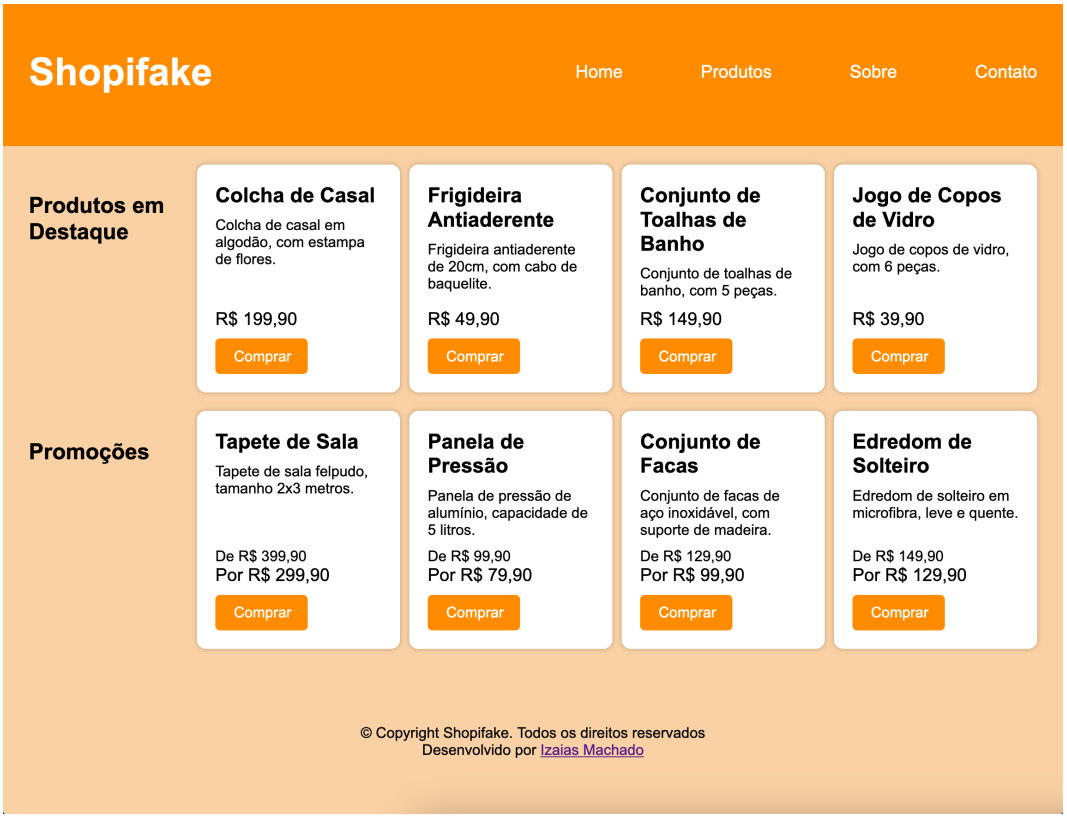
Figura 9 – Página com HTML



Por fim, a página foi populada com nomes e preços de produtos, conforme apresentado na Figura 10. Além disso, na seção de promoções foi adicionado os preços normais de

venda e o preço promocional de cada produto.

Figura 10 – Página populada com produtos



3 INSTRUÇÕES DE USO

Este capítulo visa detalhar as instruções de uso da aplicação. Portanto, foi dividido em detalhamento da versão do Node.js (Seção 3.1), instruções de instalação de dependências (Seção 3.2) e execução da aplicação (Seção 3.3).

3.1 Versão do Node.js

Esse trabalho foi feito utilizando a versão v20.9.0 do Node.js. Para garantir a compatibilidade total, garanta que você está utilizando uma versão igual ou superior, executando o comando da Listagem 3.1.1.

```
1 node --version
```

Listagem 3.1.1 – Comando para verificar versão do Node.js

3.2 Instalação de dependências

As dependências utilizadas são padrões do Node.js, entretanto, caso haja algum problema de execução é possível executar o comando da Listagem 3.2.1 na raiz do projeto para instalar essas dependências.

```
1 npm install
```

Listagem 3.2.1 – Comando para instalar dependências

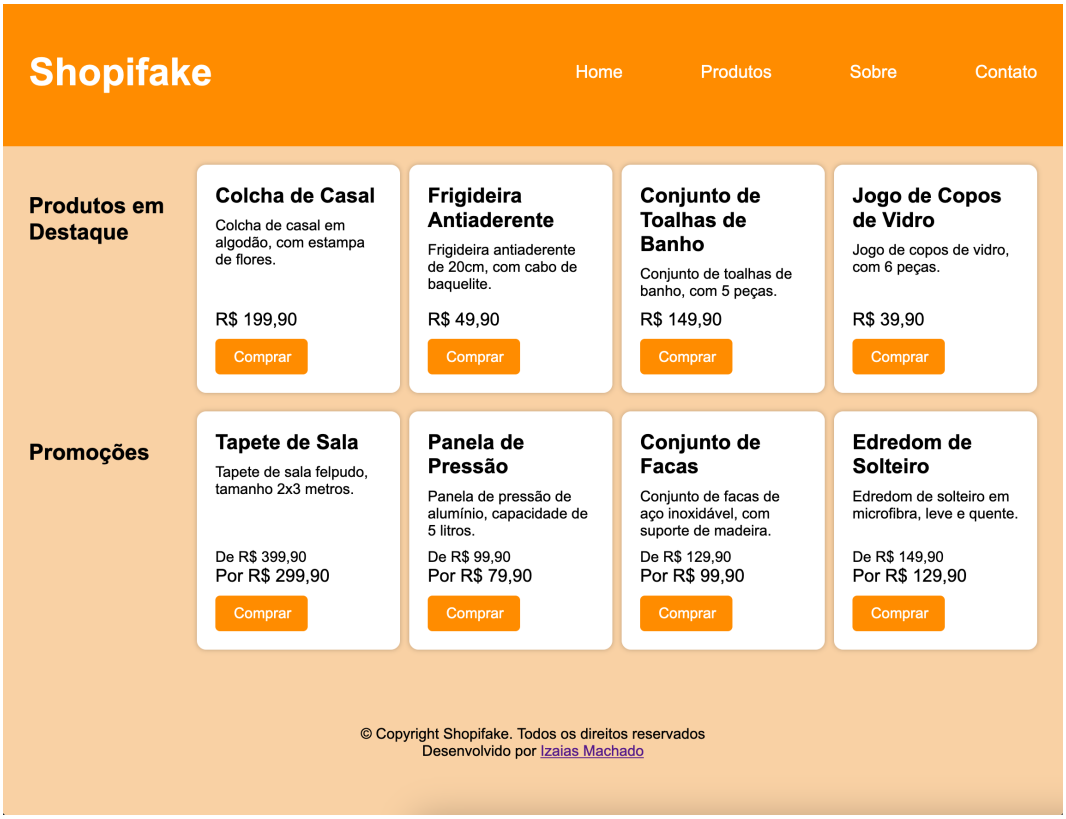
3.3 Execução da aplicação

Para executar o servidor, basta navegar até a raiz do projeto utilizando um terminal ou *prompt* de comando. E entrar com o comando da Listagem 3.3.1. Com isso, basta acessar o `http://localhost:4000` para visualizar a página principal da aplicação, conforme apresentado na Figura 11.

```
node src/server.js
```

Listagem 3.3.1 – Comando para executar o servidor

Figura 11 – Página principal da aplicação



Se por algum motivo não possa executar na porta 4000, que é a porta padrão, basta passar como variável de ambiente com o número da porta, conforme o exemplo da Listagem 3.3.2. Nesse exemplo, é necessário acessar `http://localhost:3000` para visualizar a interface de usuário.

```
PORT=3000 node src/server.js
```

Listagem 3.3.2 – Comando para executar o servidor mudando porta padrão