

## Missão Prática – Mundo 05 – Nível 05

Gilvan Pereira de Oliveira – 2023.01.53256-61197

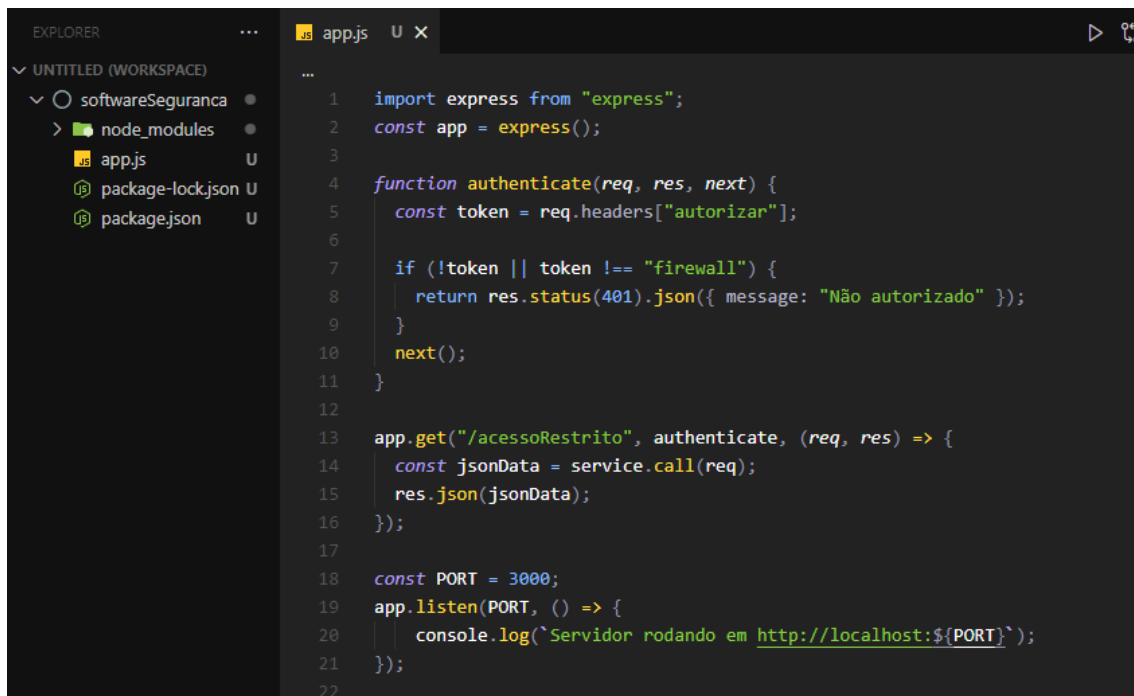
Polo Centro – São Lourenço Da Mata – PE

**RPG0034 – SOFTWARE SEM SEGURANÇA NÃO SERVE! – 9001 – 2025.1**

<https://github.com/GilvanPOliveira/FullStack/tree/main/Mundo05/softwareSeguranca>

### Micro atividade 01: Análise e Correção de Vulnerabilidades em Código

O código utiliza o framework Express para criar um servidor HTTP que processa requisições:



```
1 import express from "express";
2 const app = express();
3
4 function authenticate(req, res, next) {
5   const token = req.headers["autorizar"];
6
7   if (!token || token !== "firewall") {
8     return res.status(401).json({ message: "Não autorizado" });
9   }
10  next();
11 }
12
13 app.get("/acessoRestrito", authenticate, (req, res) => {
14   const jsonData = service.call(req);
15   res.json(jsonData);
16 });
17
18 const PORT = 3000;
19 app.listen(PORT, () => {
20   console.log(`Servidor rodando em http://localhost:${PORT}`);
21 });
22
```

- Ele define duas rotas, raiz (/) e (/acessoRestrito), que requer autenticação para acessar;
- Tratamento de Segurança com Autenticação, incluindo um middleware de autenticação que valida o acesso à rota /acessoRestrito. Ele verifica se o cabeçalho "autorizar" contém o valor "firewall";
- Retornar Mensagem Genérica e Status 401 em Caso de Acesso Não Autorizado, se o token de autenticação estiver ausente ou for inválido, o middleware retorna uma mensagem genérica ("Não autorizado") e define o status HTTP como 401 Unauthorized;
- Enviar Resposta da Requisição em Caso de Autenticação Bem-Sucedida, se o token de autenticação for válido, o middleware chama next(), permitindo que a requisição prossiga para a rota /acessoRestrito. A rota então retorna uma resposta JSON com a mensagem "Acesso permitido!".

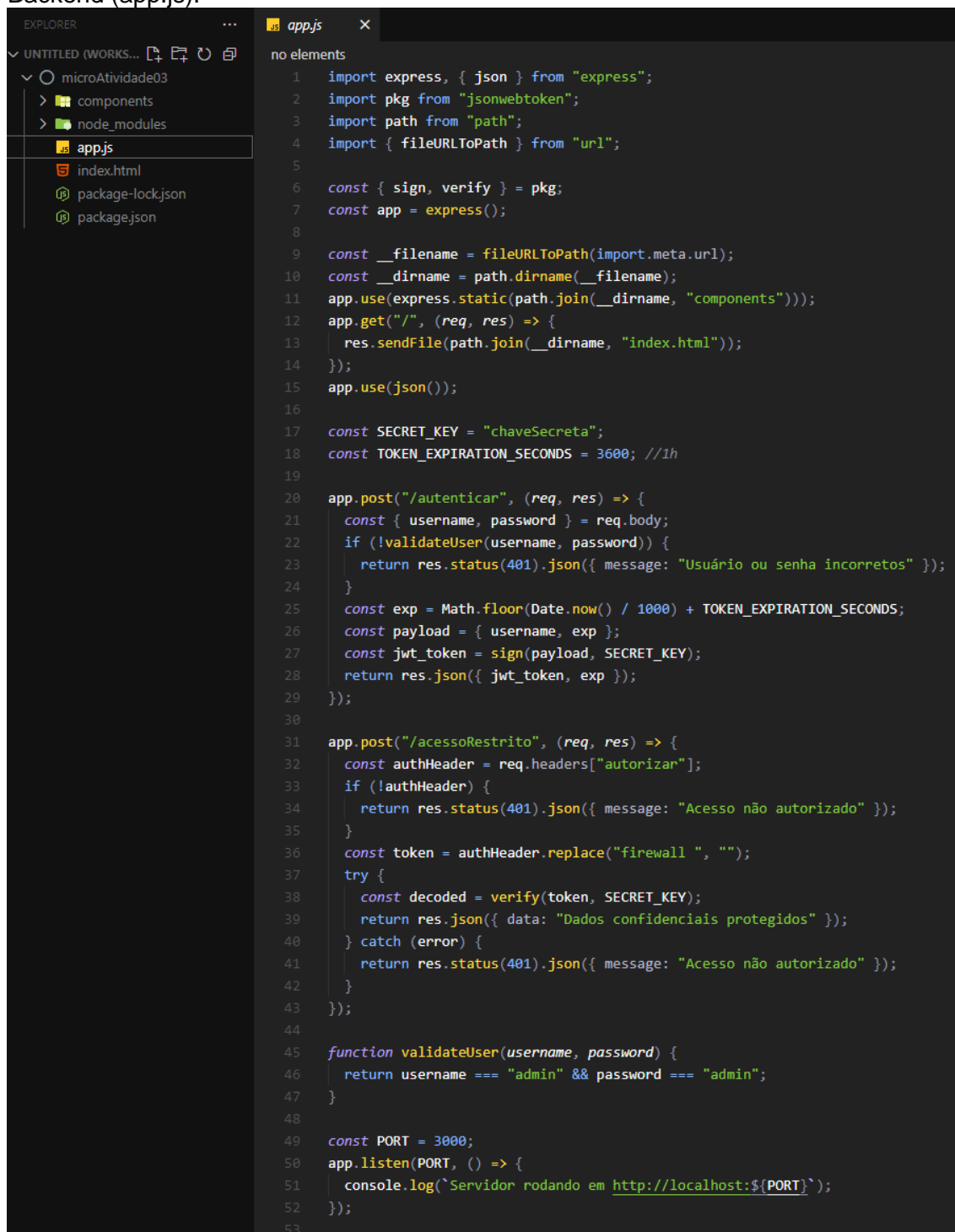
## Micro atividade 2: Descrever o tratamento de dados sensíveis e log de erros com foco em segurança

```
bd.sql  U X  ▶ 🔍 □ ...
no elements
1  MIN_PASSWORD_LENGTH = 8
2  MAX_LOGIN_ATTEMPTS = 5
3
4  password = new_password
5  username = new_username
6
7  IF USER_EXISTS(username) THEN
8      RETURN Error("Já existe usuário com esse nome.")
9  ENDIF
10
11 IF LENGTH(password) < MIN_PASSWORD_LENGTH THEN
12     RETURN Error("Senha deve ter pelo menos " + MIN_PASSWORD_LENGTH + "
13     caracteres.")
14 ENDIF
15
16 IF GET_LOGIN_ATTEMPTS(username) >= MAX_LOGIN_ATTEMPTS THEN
17     RETURN Error("Muitas tentativas inválidas. Tente novamente mais tarde.")
18 ENDIF
19
20 IF NOT LOOKUP_CREDENTIALS_IN_DATABASE(username, password) THEN
21     INCREMENT_LOGIN_ATTEMPTS(username)
22     RETURN Error("Usuário ou senha incorretos")
23 ENDIF
24
25 RESET_LOGIN_ATTEMPTS(username)
26
27 RETURN Success("Login realizado com sucesso")
```

- Verificação de usuário existente:
  - Checagem para evitar duplicidade de nomes de usuário no cadastro.
- Verificação da quantidade mínima de caracteres:
  - Foi definido um tamanho mínimo (8 caracteres) e o código verifica se a senha atende a esse requisito.
- Permissão de quaisquer caracteres na senha:
  - Removida a restrição, no código original, que limitava a senha apenas a caracteres numéricos.
- Limitação de tentativas inválidas:
  - Antes de validar as credenciais, é verificado se o número de tentativas inválidas já excedeu o limite permitido. Caso seja, o código retornará uma mensagem informando que o usuário atingiu o número máximo de tentativas.
- Mensagem de erro genérica:
  - Ao validar as credenciais, se houver qualquer erro, o código retorna a mesma mensagem: "Usuário ou senha incorretos", sem informar detalhes específicos.

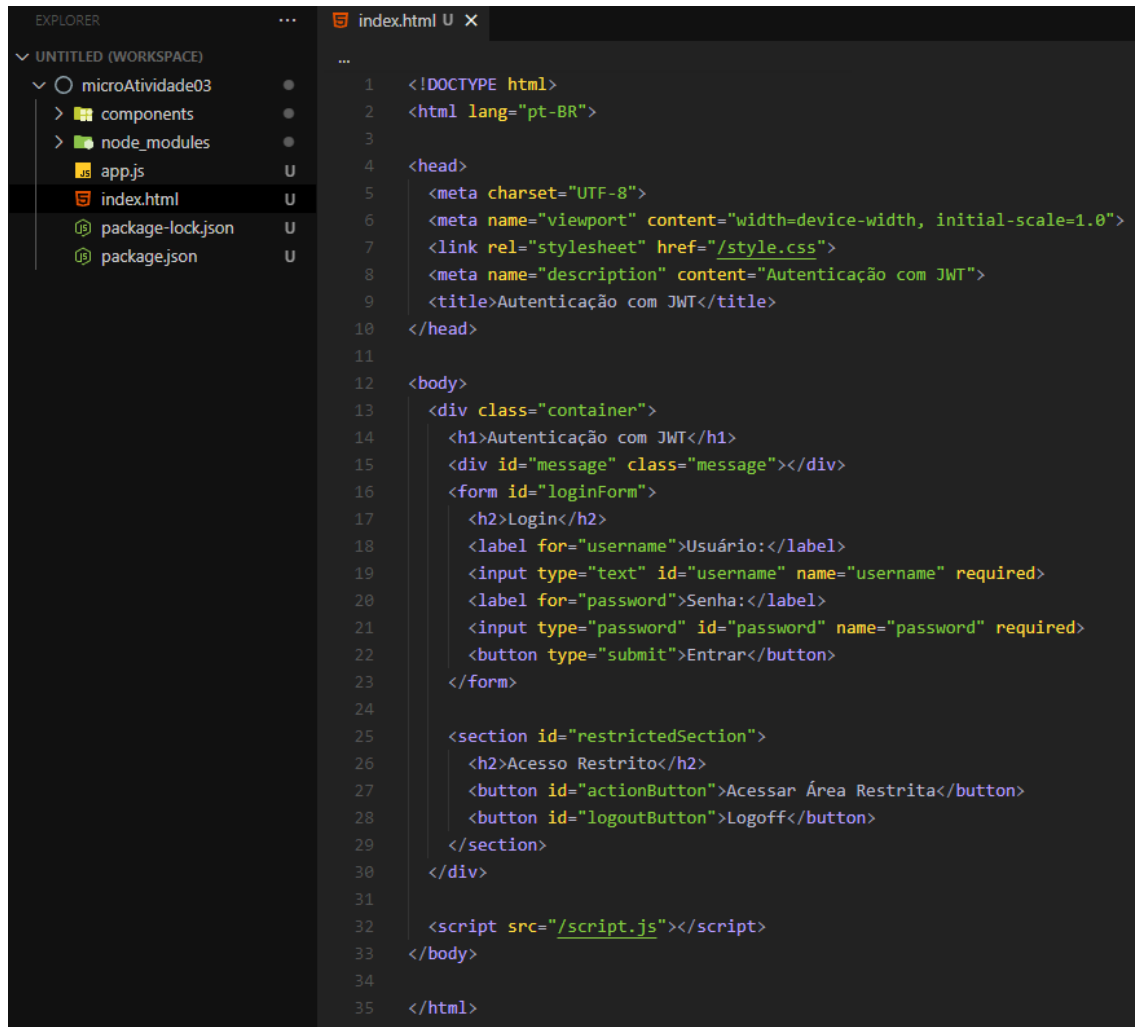
### Micro atividade 3: Descrever a prevenção de ataques de acesso não autorizado com base em tokens desprotegidos/desatualizados

Backend (app.js):



```
1 import express, { json } from "express";
2 import pkg from "jsonwebtoken";
3 import path from "path";
4 import { fileURLToPath } from "url";
5
6 const { sign, verify } = pkg;
7 const app = express();
8
9 const __filename = fileURLToPath(import.meta.url);
10 const __dirname = path.dirname(__filename);
11 app.use(express.static(path.join(__dirname, "components")));
12 app.get("/", (req, res) => {
13   res.sendFile(path.join(__dirname, "index.html"));
14 });
15 app.use(json());
16
17 const SECRET_KEY = "chaveSecreta";
18 const TOKEN_EXPIRATION_SECONDS = 3600; //1h
19
20 app.post("/autenticar", (req, res) => {
21   const { username, password } = req.body;
22   if (!validateUser(username, password)) {
23     return res.status(401).json({ message: "Usuário ou senha incorretos" });
24   }
25   const exp = Math.floor(Date.now() / 1000) + TOKEN_EXPIRATION_SECONDS;
26   const payload = { username, exp };
27   const jwt_token = sign(payload, SECRET_KEY);
28   return res.json({ jwt_token, exp });
29 });
30
31 app.post("/acessoRestrito", (req, res) => {
32   const authHeader = req.headers["autorizar"];
33   if (!authHeader) {
34     return res.status(401).json({ message: "Acesso não autorizado" });
35   }
36   const token = authHeader.replace("firewall ", "");
37   try {
38     const decoded = verify(token, SECRET_KEY);
39     return res.json({ data: "Dados confidenciais protegidos" });
40   } catch (error) {
41     return res.status(401).json({ message: "Acesso não autorizado" });
42   }
43 });
44
45 function validateUser(username, password) {
46   return username === "admin" && password === "admin";
47 }
48
49 const PORT = 3000;
50 app.listen(PORT, () => {
51   console.log(`Servidor rodando em http://localhost:${PORT}`);
52 });
53
```

Frontend (index.html):

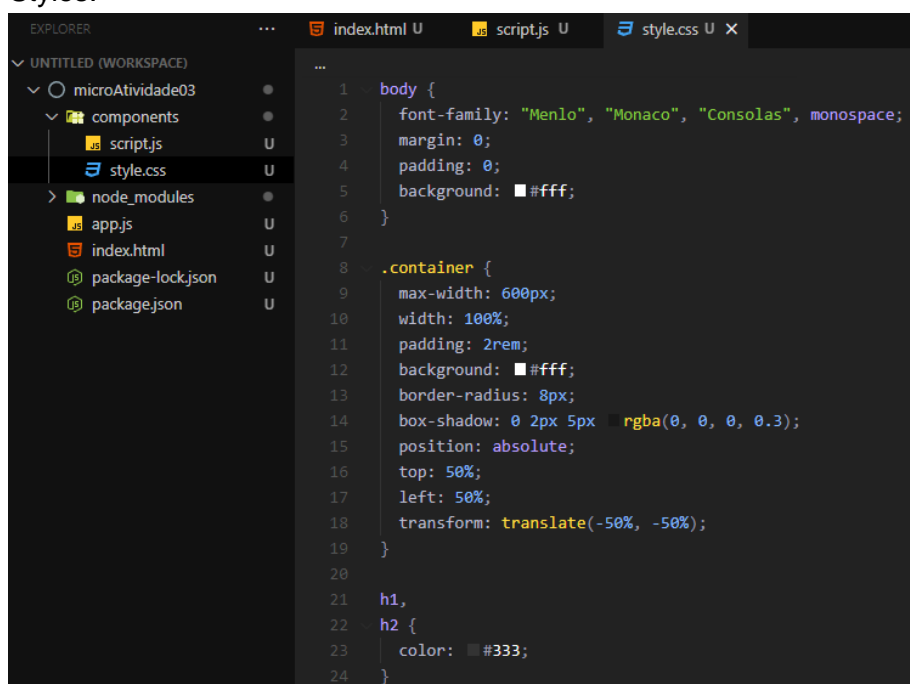


```

1  <!DOCTYPE html>
2  <html lang="pt-BR">
3
4  <head>
5    <meta charset="UTF-8">
6    <meta name="viewport" content="width=device-width, initial-scale=1.0">
7    <link rel="stylesheet" href="/style.css">
8    <meta name="description" content="Autenticação com JWT">
9    <title>Autenticação com JWT</title>
10  </head>
11
12  <body>
13    <div class="container">
14      <h1>Autenticação com JWT</h1>
15      <div id="message" class="message"></div>
16      <form id="loginForm">
17        <h2>Login</h2>
18        <label for="username">Usuário:</label>
19        <input type="text" id="username" name="username" required>
20        <label for="password">Senha:</label>
21        <input type="password" id="password" name="password" required>
22        <button type="submit">Entrar</button>
23      </form>
24
25      <section id="restrictedSection">
26        <h2>Acesso Restrito</h2>
27        <button id="actionButton">Acessar Área Restrita</button>
28        <button id="logoutButton">Logoff</button>
29      </section>
30    </div>
31
32    <script src="/script.js"></script>
33  </body>
34
35  </html>

```

Styles:



```

1  body {
2    font-family: "Menlo", "Monaco", "Consolas", monospace;
3    margin: 0;
4    padding: 0;
5    background: #fff;
6  }
7
8  .container {
9    max-width: 600px;
10   width: 100%;
11   padding: 2rem;
12   background: #fff;
13   border-radius: 8px;
14   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);
15   position: absolute;
16   top: 50%;
17   left: 50%;
18   transform: translate(-50%, -50%);
19 }
20
21 h1,
22 h2 {
23   color: #333;
24 }

```

## Frontend (script.js):

```

EXPLORER
  ...
  index.html U
  script.js U X
  ...
  UNTITLED (WORKSPACE)
  microAtividade03
    components
      script.js
      style.css
    node_modules
    app.js
    index.html
    package-lock.json
    package.json
  ...

1  const messageDiv = document.getElementById("message");
2
3  function showMessage(text, type) {
4    messageDiv.textContent = text;
5    messageDiv.className = "message " + (type === "error" ? "error" : "success");
6    messageDiv.style.display = "block";
7  }
8
9  // Evento de Login
10 document.getElementById("loginForm").addEventListener("submit", function (e) {
11   e.preventDefault();
12   messageDiv.style.display = "none";
13
14   const username = document.getElementById("username").value;
15   const password = document.getElementById("password").value;
16   const data = { username, password };
17
18   fetch("http://localhost:3000/autenticar", {
19     method: "POST",
20     headers: { "Content-Type": "application/json" },
21     body: JSON.stringify(data),
22   })
23     .then((response) => response.json())
24     .then((json) => {
25       if (json.jwt_token) {
26         localStorage.setItem("token", json.jwt_token);
27         localStorage.setItem("tokenExp", json.exp);
28         showMessage("Login efetuado com sucesso!", "success");
29       } else {
30         showMessage(json.message || "Erro no login.", "error");
31       }
32     })
33     .catch((err) => {
34       console.error(err);
35       showMessage("Erro ao se conectar com o servidor.", "error");
36     });
37 });
38
39 // Evento para acesso restrito
40 document.getElementById("actionButton").addEventListener("click", function () {
41   messageDiv.style.display = "none";
42   const token = localStorage.getItem("token");
43   const tokenExp = parseInt(localStorage.getItem("tokenExp"), 10);
44   const now = Math.floor(Date.now() / 1000);
45
46   if (!token || now >= tokenExp) {
47     showMessage(
48       "Token expirado ou inexistente, por favor faça o login novamente.",
49       "error"
50     );
51     return;
52   }
53
54   fetch("http://localhost:3000/acessoRestrito", {
55     method: "POST",
56     headers: {
57       "Content-Type": "application/json",
58       autorizar: `firewall ${token}`,
59     },
60   })
61     .then((response) => response.json())
62     .then((json) => {
63       showMessage(json.data, "success");
64     })
65     .catch((err) => {
66       console.error(err);
67       showMessage("Erro ao acessar a área restrita.", "error");
68     });
69 });
70
71 // Evento para Logout
72 document.getElementById("logoutButton").addEventListener("click", function () {
73   localStorage.removeItem("token");
74   localStorage.removeItem("tokenExp");
75   showMessage("Você foi deslogado com sucesso.", "success");
76 });

```

Backend (app.js):

1. Geração do Token com Claim "exp":
  - O endpoint /autenticar valida as credenciais (neste caso, usuário e senha "admin");
  - Em caso de sucesso, calcula o timestamp de expiração (exp) e inclui-o no payload do token JWT (determinado por 3600s, ou seja, 1 hora);
  - O token e o timestamp de expiração são retornados no JSON de resposta.
2. Validação do Token e Expiração:
  - endpoint /acessoRestrito extrai o token do cabeçalho "autorizar", remove o prefixo "firewall " e utiliza verify (token, chaveSecreta) para validar o token;
  - Caso a validação falhe, o bloco catch retorna uma mensagem genérica, atendendo ao requisito de não revelar detalhes sobre a causa da falha.
3. Organização dos Arquivos Estáticos:
  - Todos os arquivos estáticos (como CSS e JS) estão na pasta componentes;
  - O index.html é servido via res.sendFile, e reside na raiz do projeto.
4. Validação de Credenciais:
  - A função validateUser compara os valores com "admin". Essa verificação simples é suficiente para o que foi solicitado nesta micro atividade.

Frontend (index.html e script.js):

1. Recebimento e Armazenamento do Token:
  - Ao efetuar o login, a requisição POST para /autenticar armazena tanto o jwt\_token quanto o exp no localStorage.
2. Validação do Token Antes da Requisição:
  - Antes de chamar o endpoint /acessoRestrito, o script verifica se o token existe e se o timestamp atual (em segundos, 3600) é menor que o valor armazenado;
  - Assim, se o token estiver expirado, a requisição não é realizada e uma mensagem de erro é exibida.
3. Exibição de Mensagens e Logoff:
  - A função showMessage é utilizada para exibir feedback na página;
  - O botão "Logoff" remove os itens "token" e "tokenExp" do localStorage, deslogando o usuário, já que ao logar a credencial permanece ativa por 1 hora.

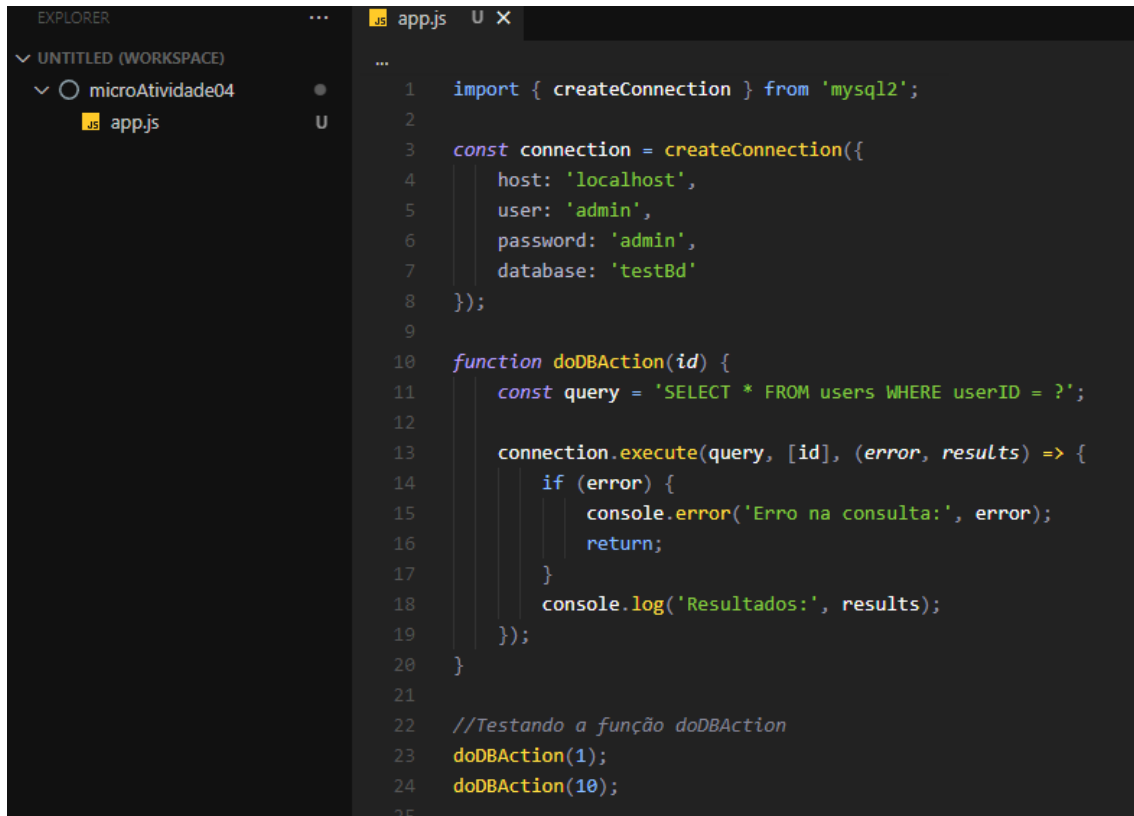
Requisitos solicitados:

- **Backend:**
  - Gera um token JWT com o claim "exp";
  - Valida o token e sua expiração, retornando um erro genérico se o token for inválido ou expirado;
  - As credenciais são validadas e o token é retornado ao frontend.
- **Frontend:**
  - Armazena o token e o timestamp de expiração no localStorage;
  - Verifica a validade do token antes de realizar uma requisição ao backend;
  - Possui botões para login, acesso restrito e logoff, com exibição adequada de mensagens.

### Micro atividade 4: Descrever o tratamento de SQL Injection em códigos-fonte

A principal falha encontrada, é que o parâmetro id é concatenado diretamente na query, permitindo que um invasor injete comandos maliciosos. Para evitar esse problema, é recomendado usar declarações preparadas ou queries parametrizadas, que tratam os parâmetros de forma segura.

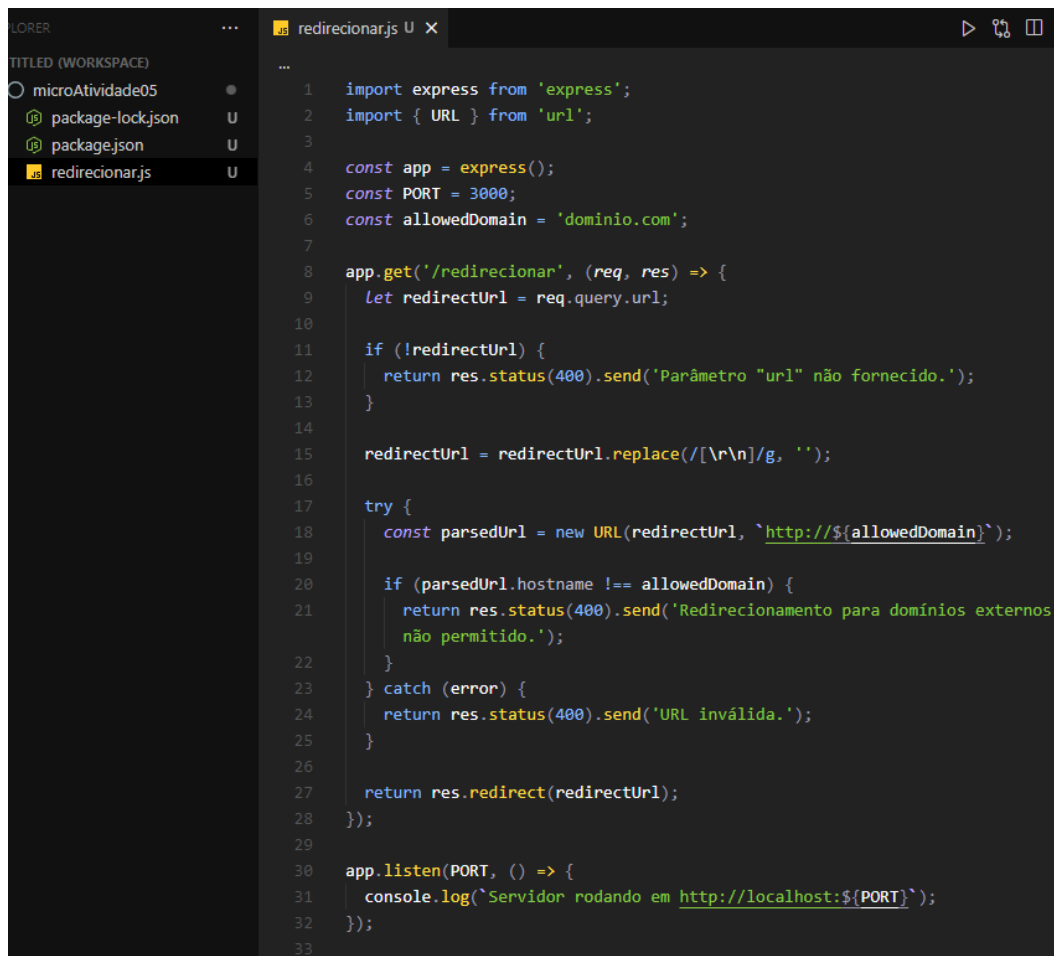
Segue um exemplo utilizando o módulo mysql2, que suporta prepared statements:



```
1 import { createConnection } from 'mysql2';
2
3 const connection = createConnection({
4   host: 'localhost',
5   user: 'admin',
6   password: 'admin',
7   database: 'testBd'
8 });
9
10 function doDBAction(id) {
11   const query = 'SELECT * FROM users WHERE userID = ?';
12
13   connection.execute(query, [id], (error, results) => {
14     if (error) {
15       console.error('Erro na consulta:', error);
16       return;
17     }
18     console.log('Resultados:', results);
19   });
20 }
21
22 //Testando a função doDBAction
23 doDBAction(1);
24 doDBAction(10);
25
```

- Prepared Statements (declarações preparadas):
  - A query SQL foi modificada para usar um (?) no lugar da concatenação direta do valor de id;
  - O valor de id é passado como um parâmetro seguro para a função connection.execute.
- Parameterized Queries (queries parametrizadas):
  - O método execute do mysql2 garante que o valor de id seja tratado como um parâmetro seguro, evitando que ele seja interpretado como parte da query SQL.
- Segurança:
  - Mesmo que um atacante tente enviar um valor malicioso como ' OR '1'='1, ele será tratado como um valor literal e não como parte da query SQL.

## Micro atividade 5: Descrever o tratamento de CRLF Injection em códigos-fonte



```
1 import express from 'express';
2 import { URL } from 'url';
3
4 const app = express();
5 const PORT = 3000;
6 const allowedDomain = 'dominio.com';
7
8 app.get('/redirecionar', (req, res) => {
9   let redirectUrl = req.query.url;
10
11   if (!redirectUrl) {
12     return res.status(400).send('Parâmetro "url" não fornecido.');
```

1. Recebimento da URL: A rota /redirecionar captura o parâmetro url enviado via query string. Por exemplo, na URL `http://dominio.com/redirect?url=/pagina`, o valor /pagina é extraído como a URL de destino;
2. Sanitização: Antes de qualquer processamento, o código remove todos os caracteres de quebra de linha (CR – \r e LF – \n) utilizando `redirectUrl.replace(/[\\r\\n]/g, '')`. Essa etapa é crucial para evitar a injeção de cabeçalhos HTTP adicionais por meio de caracteres maliciosos;
3. Validação do Domínio:
  - O construtor `new URL(redirectUrl, base)` é empregado para criar um objeto URL. Se o valor fornecido for uma URL relativa, ele será resolvido com base no domínio permitido (que foi denominada: `dominio.com`);
  - Em seguida, é verificado se o hostname da URL resultante é exatamente igual ao domínio autorizado (definido em `allowedDomain`). Se não for, o redirecionamento é impedido, evitando que o usuário seja direcionado para um domínio externo não autorizado;
4. Redirecionamento Seguro: Se a URL estiver devidamente sanitizada e validada (ou seja, pertence ao domínio permitido ou é uma URL relativa que se resolve para ele), o código executa o redirecionamento chamando `res.redirect(redirectUrl)`. Dessa forma, o redirecionamento ocorre somente para destinos seguros.