at_S7_A3_SL6_backend

Roteiro de atividade prática

Nome: _____ Turma: _____

Título da atividade: implementação de WebSockets e autenticação/autorização no *back-end*

Pontos principais do conteúdo:

- **WebSockets para comunicação em tempo real:** WebSockets são uma tecnologia que permite a comunicação bidirecional e em tempo real entre cliente e servidor. Ao contrário de uma requisição HTTP tradicional, em que o cliente faz uma solicitação e o servidor responde, o WebSocket cria uma conexão persistente que permite a troca contínua de dados sem a necessidade de repetidas requisições HTTP. Isso é ideal para aplicações como chats, jogos em tempo real ou notificações instantâneas.
- **Autenticação e autorização:** a autenticação é o processo de validar a identidade do usuário, geralmente por meio de credenciais como nome de usuário e senha. A autorização verifica se o usuário autenticado tem permissão para acessar recursos específicos. Quando combinados com WebSockets, é importante garantir que os usuários autenticados possam acessar e enviar mensagens em canais autorizados.

Objetivos

- **Implementar WebSockets** para permitir a comunicação em tempo real entre o cliente e o servidor em uma aplicação *back-end*.
- **Adicionar autenticação e autorização** ao sistema, garantindo que apenas usuários autenticados possam se conectar ao WebSocket e que certos canais ou funcionalidades estejam disponíveis apenas para usuários com permissões específicas (papéis como "admin" ou "user").

- **Proteger a comunicação em tempo real**, integrando autenticação com *tokens* JWT e autorização com base no papel do usuário.

Contexto:

Você está desenvolvendo uma aplicação de chat em tempo real para um sistema de suporte ao cliente. Os usuários autenticados podem se conectar ao WebSocket e enviar mensagens instantâneas para os atendentes, enquanto os administradores (admin) têm a permissão de monitorar todas as salas de chat.

Sua tarefa é:

- implementar **WebSockets** para permitir a comunicação em tempo real entre clientes e atendentes;
- implementar um sistema de **autenticação** com *tokens* JWT, permitindo que apenas usuários autenticados se conectem ao WebSocket;
- garantir que **apenas administradores** tenham permissão para acessar certos canais de chat, utilizando um sistema de autorização baseado em papéis (roles).

Tarefa:

1. **Configurar a autenticação com JWT:**

- Implemente uma rota de login que valide as credenciais do usuário e retorne um token JWT com o papel (*role*) "admin" ou "user".
- O *token* JWT será utilizado para autenticar o usuário no WebSocket.

2. **Implementar WebSockets:**

- Configure o WebSocket no servidor para permitir conexões em tempo real.
- Crie um *middleware* para validar o *token* JWT quando o cliente se conectar ao WebSocket, garantindo que apenas usuários autenticados possam iniciar a comunicação.

3. **Proteger o acesso a canais com base em papéis:**
- Garanta que certas funcionalidades do WebSocket (como monitorar salas de chat) sejam acessíveis apenas para usuários com o papel "admin". Por exemplo, usuários comuns podem apenas enviar e receber mensagens em suas salas, enquanto administradores podem acessar todas as salas de chat.

Exemplo de fluxo:

- **POST /login:** o usuário faz login e recebe um *token* JWT.
- **WebSocket conexão:** o cliente usa o *token* JWT para se conectar ao WebSocket. O servidor valida o *token*.
- **Acesso autorizado:** usuários autenticados podem trocar mensagens. Administradores podem monitorar todas as salas de chat.

Código de exemplo (Node.js com WebSockets e JWT):

```
const express = require('express');
const jwt = require('jsonwebtoken');
const http = require('http');
const WebSocket = require('ws');
const app = express();

const SECRET_KEY = 'my_secret_key';
const users = [
   { id: 1, username: 'user1', password: 'password1', role: 'user' },
   { id: 2, username: 'admin', password: 'adminpass', role: 'admin' }
];

// Função para gerar token JWT
function generateToken(user) {
   return jwt.sign({ id: user.id, role: user.role }, SECRET_KEY, { expiresIn: '1h' });
}

// Middleware para autenticação do WebSocket
```

```
function authenticateWebSocket(token, cb) {
  if (!token) return cb('Token não fornecido');
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return cb('Token inválido');
    cb(null, user);
  });
}

// Rota de login
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find(u => u.username === username && u.password ===
password);

  if (user) {
    const token = generateToken(user);
    res.json({ token });
  } else {
    res.sendStatus(401);
  }
});

const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

// Manipulando a conexão WebSocket
wss.on('connection', (ws, req) => {
  const token = req.url.split('?token=')[1];

  authenticateWebSocket(token, (err, user) => {
    if (err) {
      ws.close();
      return;
    }

    ws.on('message', (message) => {
```

```
        if (user.role === 'admin') {
            console.log('Admin enviou uma mensagem:', message);
        } else {
            console.log('Usuário enviou uma mensagem:', message);
        }
    });

    ws.send(`Bem-vindo, ${user.role}!`);
  });
});

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

Perguntas para responder

1. Como a autenticação e a autorização podem ser implementadas em WebSockets para garantir que apenas usuários autorizados acessem os canais?

2. Quais são os principais benefícios de utilizar WebSockets em comparação com requisições HTTP tradicionais para comunicação em tempo real?

3. Como você pode diferenciar o acesso a funcionalidades no WebSocket com base no papel (*role*) do usuário?