

Material de apoio

Site: [Geração Tech](#)
Curso: Formação em Desenvolvedor Web - Online
Livro: Material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: segunda-feira, 12 ago. 2024, 22:12

Índice

1. Autenticação JWT

1.1. Vídeo Aula

2. Configurando JWT em um Projeto Node.js

2.1. Vídeo Aula

3. Implementando Expiração de Tokens JWT

3.1. Vídeo Aula

4. Explicação sobre JWT e Configuração de Expiração de Tokens

4.1. Vídeo Aula

1. Autenticação JWT

Introdução ao JWT

Olá pessoal, tudo bem? Meu nome é Márcio Ferreira e hoje vamos iniciar o curso de autenticação JWT (JSON Web Token).

Neste curso, vamos entender o que é JWT, como ele funciona e como utilizá-lo para controlar a autenticação de usuários em aplicações web.

O que é JWT?

JWT significa JSON Web Token. Ele é um padrão aberto (RFC 7519) que define uma forma compacta e segura de transmitir informações entre as partes como um objeto JSON. Essas informações podem ser verificadas e confiáveis porque são assinadas digitalmente.

Como Funciona o JWT?

O JWT é composto por três partes:

1. **Header (Cabeçalho):** Contém o tipo do token (JWT) e o algoritmo de criptografia (por exemplo, HMAC SHA256 ou RSA).
2. **Payload (Corpo):** Contém as declarações (claims), que são as informações que queremos transmitir (por exemplo, informações do usuário).
3. **Signature (Assinatura):** É criada a partir do header, do payload e de uma chave secreta usando o algoritmo especificado no header.

Essas três partes são codificadas em Base64URL e concatenadas com pontos, formando o token final.

Processo de Autenticação com JWT

1. **Login:** O usuário envia suas credenciais (login e senha) para o servidor.
2. **Validação:** O servidor verifica as credenciais e, se forem válidas, gera um JWT e o retorna ao usuário.
3. **Armazenamento do Token:** O cliente (front-end) armazena o token, geralmente no localStorage ou sessionStorage.
4. **Acesso às Rotas Protegidas:** Para acessar rotas protegidas, o cliente envia o token no cabeçalho da requisição (Authorization: Bearer TOKEN).
5. **Validação do Token:** O servidor valida o token. Se for válido, a requisição é processada; caso contrário, o acesso é negado.

Analogia com Controle de Acesso

Podemos comparar o JWT com o controle de acesso a um prédio:

- **Recepção:** Onde você se identifica (login e senha).
- **Recepção Válida:** A recepção valida suas informações e gera um crachá (JWT).
- **Acesso:** Você usa o crachá para acessar diferentes áreas do prédio. O crachá tem uma validade e, após expirar, você precisa se identificar novamente.

Exemplo Prático

Vamos criar um exemplo prático de autenticação usando JWT.

Passo 1: Configuração do Projeto

1. **Criar a Pasta do Projeto:**


```
bash
```

 Copiar código

```
mkdir aula-jwt cd aula-jwt npm init -y
```

2. **Instalar Dependências:**

```
bash
```

 Copiar código

```
npm install express jsonwebtoken bcryptjs body-parser
```

3. Estrutura de Pastas:


go

 Copiar código

```
aula-jwt ├── node_modules ├── src ├── controllers ├── middlewares ├── models ├── routes ├── index.js ├──  
package.json └── package-lock.json
```

Passo 2: Configuração do Servidor Express

javascript

 Copiar código

```
// src/index.js const express = require('express'); const bodyParser = require('body-parser'); const userRoutes =  
require('./routes/userRoutes'); const app = express(); const PORT = 3000; app.use(bodyParser.json()); app.use('/users',  
userRoutes); app.listen(PORT, () => { console.log(`Server is running on port ${PORT}`); });
```

Passo 3: Configuração das Rotas de Usuário


javascript

 Copiar código

```
// src/routes/userRoutes.js const express = require('express'); const { register, login } =  
require('../controllers/userController'); const router = express.Router(); router.post('/register', register);  
router.post('/login', login); module.exports = router;
```

Passo 4: Controlador de Usuário

javascript

 Copiar código

```
// src/controllers/userController.js const jwt = require('jsonwebtoken'); const bcrypt = require('bcryptjs'); const users =  
[]; // Temporário: armazenar usuários na memória const register = (req, res) => { const { username, password } = req.body;  
const hashedPassword = bcrypt.hashSync(password, 8); users.push({ username, password: hashedPassword });  
res.status(201).send({ message: 'User registered successfully!' }); }; const login = (req, res) => { const { username,  
password } = req.body; const user = users.find(u => u.username === username); if (!user || !bcrypt.compareSync(password,  
user.password)) { return res.status(401).send({ message: 'Invalid credentials!' }); } const token = jwt.sign({ id:  
user.username }, 'secret_key', { expiresIn: '1h' }); res.send({ auth: true, token }); }; module.exports = { register, login  
};
```

Passo 5: Middleware de Autenticação

javascript

 Copiar código

```
// src/middlewares/auth.js const jwt = require('jsonwebtoken'); const verifyToken = (req, res, next) => { const token =  
req.headers['authorization']; if (!token) { return res.status(403).send({ auth: false, message: 'No token provided.' }); }  
jwt.verify(token, 'secret_key', (err, decoded) => { if (err) { return res.status(500).send({ auth: false, message: 'Failed  
to authenticate token.' }); } req.userId = decoded.id; next(); }); }; module.exports = verifyToken;
```

Passo 6: Protegendo Rotas com Middleware

javascript


 Copiar código

```
// src/routes/userRoutes.js const express = require('express'); const { register, login } =
require('../controllers/userController'); const verifyToken = require('../middlewares/auth'); const router =
express.Router(); router.post('/register', register); router.post('/login', login); router.get('/me', verifyToken, (req,
res) => { res.status(200).send({ message: `User ID: ${req.userId}` }); }); module.exports = router;
```

Testando a Aplicação

1. **Registro de Usuário:** Envie uma requisição POST para `/users/register` com JSON:

json

 Copiar código

```
{ "username": "testuser", "password": "testpassword" }
```

2. **Login de Usuário:** Envie uma requisição POST para `/users/login` com JSON:

json

 Copiar código

```
{ "username": "testuser", "password": "testpassword" }
```

3. **Acesso à Rota Protegida:** Envie uma requisição GET para `/users/me` com o cabeçalho `Authorization: Bearer TOKEN`.

Conclusão

Implementamos uma autenticação básica usando JWT em uma aplicação Node.js com Express. Aprendemos a registrar, fazer login e proteger rotas usando tokens JWT.

Nos próximos passos, podemos explorar como manejar a renovação de tokens, logout e outras melhores práticas de segurança.

Espero que tenham gostado dessa introdução e, em breve, continuaremos com mais detalhes e funcionalidades avançadas. Até a próxima!

1.1. Vídeo Aula

dia 41 jwt vd1 converted



2. Configurando JWT em um Projeto Node.js

Olá pessoal, tudo bem? Vamos dar continuidade à nossa aula sobre JWT (JSON Web Token).


Hoje, vamos configurar um projeto simples para entender como funciona o JWT na prática.

Passo a Passo para Configurar JWT

1. Configurar o Projeto

Primeiro, vamos criar um novo projeto Node.js e instalar as dependências necessárias. Abra o terminal e execute os seguintes comandos:

```
bash
```

 Copiar código


```
mkdir aula-jwt cd aula-jwt npm init -y
```

Este comando cria um arquivo `package.json` com as configurações básicas do projeto.

2. Instalar a Biblioteca JWT

Instale a biblioteca `jsonwebtoken`:

```
bash
```

 Copiar código

```
npm install jsonwebtoken
```


3. Criar o Arquivo do Servidor

Crie um arquivo chamado `server.js` ou `index.js` na raiz do projeto. Este arquivo será o nosso servidor onde vamos configurar o JWT.

4. Implementar a Geração e Verificação do Token

No arquivo `server.js`, vamos implementar a geração e verificação do JWT. O código ficará assim:

```
javascript
```

 Copiar código

```
// server.js const jwt = require('jsonwebtoken'); // Dados do usuário (simulação) const dados = { nome: 'Márcio', email: 'marcio@gt.com.br' }; // Chave secreta para assinar o token const chaveSecreta = 'minha_chave_secreta'; // Função para gerar o token const gerarToken = (dados) => { return jwt.sign(dados, chaveSecreta); }; // Função para verificar o token const verificarToken = (token) => { try { const decoded = jwt.verify(token, chaveSecreta); console.log('Token válido:', decoded); } catch (error) { console.error('Erro ao verificar token:', error.message); } }; // Gerar um token const token = gerarToken(dados); console.log('Token gerado:', token); // Verificar o token verificarToken(token); // Simulação de alteração no token const tokenAlterado = token.slice(0, -1) + 'a'; verificarToken(tokenAlterado);
```

5. Executar o Servidor

Para executar o servidor, use o comando:

```
bash
```


 Copiar código

```
node server.js
```

Explicação do Código

1. Importação da Biblioteca JWT:


javascript

 Copiar código

```
const jwt = require('jsonwebtoken');
```

2. Dados do Usuário:


javascript

 Copiar código

```
const dados = { nome: 'Márcio', email: 'marcio@gt.com.br' };
```

3. Chave Secreta:

javascript

 Copiar código

```
const chaveSecreta = 'minha_chave_secreta';
```

4. Função para Gerar o Token:


javascript

 Copiar código

```
const gerarToken = (dados) => { return jwt.sign(dados, chaveSecreta); };
```

5. Função para Verificar o Token:

javascript

 Copiar código

```
const verificarToken = (token) => { try { const decoded = jwt.verify(token, chaveSecreta); console.log('Token válido:', decoded); } catch (error) { console.error('Erro ao verificar token:', error.message); } };
```

6. Gerar e Verificar o Token:

javascript

 Copiar código

```
const token = gerarToken(dados); console.log('Token gerado:', token); verificarToken(token);
```

7. Simulação de Alteração no Token:

javascript

 Copiar código

```
const tokenAlterado = token.slice(0, -1) + 'a'; verificarToken(tokenAlterado);
```

Tratamento de Erros

Para tratar possíveis erros ao verificar o token, utilizamos `try...catch`:

javascript

 Copiar código


```
try { const decoded = jwt.verify(token, chaveSecreta); console.log('Token válido:', decoded); } catch (error) { console.error('Erro ao verificar token:', error.message); }
```

Tarefas para Prática

1. Instalar as Dependências:

- Crie um novo projeto com `npm init -y`.
- Instale a biblioteca `jsonwebtoken` com `npm install jsonwebtoken`.

2. Implementar o Código:

- Crie o arquivo `server.js` e implemente o código fornecido.
- Execute o servidor com `node server.js` para ver o token gerado e a verificação do token.

3. Explorar a Biblioteca JWT:

- Experimente alterar os dados do usuário e a chave secreta.
- Tente alterar o token manualmente e observe o comportamento da verificação.

Pratiquem bastante e até a próxima aula!

2.1. Vídeo Aula

dia 41 jwt vd2 converted



3. Implementando Expiração de Tokens JWT

Olá pessoal, vamos dar continuidade à nossa aula sobre JWT, agora trabalhando com a expiração dos tokens.

Criamos tokens, mas eles não têm prazo de validade. Vamos ver como configurar isso.


1. Criar um Arquivo de Expiração

Primeiro, vamos criar um arquivo chamado `expiracao.js` para trabalhar com a expiração dos tokens.

1.1 Trabalhando com `Date` no JavaScript

Vamos usar o objeto `Date` para obter o tempo atual em milissegundos desde 1 de janeiro de 1970 (Unix Epoch).


javascript

 Copiar código

```
// expiracao.js const dateNow = Date.now(); // Obter o tempo atual em milissegundos console.log(dateNow);
```

Execute o arquivo para ver o resultado:

bash

 Copiar código

```
node expiracao.js
```

Este valor representa o número de milissegundos desde 1 de janeiro de 1970 até o momento atual. Vamos converter isso em segundos e arredondar:

javascript

 Copiar código

```
// expiracao.js const dateNow = Date.now(); const segundosAteAqui = Math.floor(dateNow / 1000); // Converter milissegundos em segundos e arredondar console.log(segundosAteAqui);
```

1.2 Calculando a Expiração

Vamos calcular a expiração adicionando uma hora ao tempo atual:

javascript

 Copiar código

```
// expiracao.js const dateNow = Date.now(); const segundosAteAqui = Math.floor(dateNow / 1000); const umaHoraEmSegundos = 60 * 60; const expiracao = segundosAteAqui + umaHoraEmSegundos; console.log(`Segundos até aqui: ${segundosAteAqui}`); console.log(`Expiração (em segundos): ${expiracao}`);
```

2. Gerando Tokens JWT com Expiração

Vamos criar um arquivo `gerarToken.js` para gerar tokens com expiração.

2.1 Configurando a Expiração no Token

Crie o arquivo `gerarToken.js`:

javascript

 Copiar código


```
// gerarToken.js const jwt = require('jsonwebtoken'); // Dados do usuário (simulação) const dados = { nome: 'Márcio', email: 'marcio@gt.com.br' }; // Chave secreta para assinar o token const chaveSecreta = 'minha_chave_secreta'; // Configurar a expiração para 1 hora const expiracao = Math.floor(Date.now() / 1000) + (60 * 60); // 1 hora // Função para gerar o token
```

```
const gerarToken = (dados) => { return jwt.sign(dados, chaveSecreta, { expiresIn: expiracao }); }; // Gerar o token const token = gerarToken(dados); console.log('Token gerado:', token);
```

2.2 Executar o Script para Gerar o Token

Execute o script:

bash

 Copiar código

```
node gerarToken.js
```

3. Verificando a Expiração do Token

Vamos criar um arquivo `server.js` para verificar a expiração do token.

3.1 Implementar a Verificação do Token

Crie o arquivo `server.js`:

javascript


 Copiar código

```
// server.js const jwt = require('jsonwebtoken'); // Chave secreta para verificar o token const chaveSecreta = 'minha_chave_secreta'; // Token gerado (substitua pelo token gerado no `gerarToken.js`) const token = 'SEU_TOKEN_AQUI'; try { const decoded = jwt.verify(token, chaveSecreta); console.log('Token válido:', decoded); } catch (error) { console.error('Erro ao verificar token:', error.message); }
```

3.2 Testar a Verificação do Token

Execute o script:

bash


 Copiar código

```
node server.js
```

3.3 Configurando a Expiração para Testes

Podemos ajustar a expiração para 20 segundos para testar mais facilmente:

javascript

 Copiar código

```
// gerarToken.js (ajuste) const expiracao = Math.floor(Date.now() / 1000) + 20; // 20 segundos
```

Gere o token novamente e teste a verificação no `server.js`.

Conclusão

Agora temos um exemplo completo de como gerar tokens JWT com expiração e como verificar esses tokens, incluindo o tratamento de erros.

Certifique-se de praticar gerando e verificando tokens para entender bem o fluxo de autenticação e expiração.

Pratiquem bastante e até a próxima aula!

3.1. Vídeo Aula

dia 41 jwt vd3 converted



4. Explicação sobre JWT e Configuração de Expiração de Tokens

O que é JWT?

JWT (JSON Web Token) é um padrão de autenticação amplamente utilizado atualmente. Ele é composto por três partes principais: o header, o payload e a assinatura.

1. **Header:** Contém informações sobre o tipo de token e o algoritmo de criptografia utilizado.
2. **Payload:** Contém as informações do usuário e outras informações relevantes.
3. **Assinatura:** Garante que o token não foi alterado. É gerada a partir do header, do payload e de uma chave secreta.

A criptografia do JWT transforma essas informações em um hash, que é enviado ao cliente. Quando o cliente faz uma solicitação ao back-end, o token é enviado para validação.

Visualizando um JWT

Você pode usar o site [JWT.io](https://jwt.io) para visualizar e verificar tokens JWT. O site permite que você veja as três partes do token de forma clara:

- O hash criptografado
- O JSON descriptografado, mostrando o header, o payload e a assinatura.

Isso é útil para entender como o JWT funciona e quais informações estão contidas no token.

Expiração de Tokens

Os tokens JWT devem ter um tempo de expiração para garantir a segurança. Não faz sentido uma aplicação manter um token válido indefinidamente. Vamos configurar um token para expirar em 10 minutos, por exemplo.

Implementação

1. Criar um Token com Expiração

Vamos criar um arquivo chamado `gerarToken.js` para gerar tokens com expiração.

javascript

 Copiar código

```
// gerarToken.js
const jwt = require('jsonwebtoken'); // Dados do usuário (simulação)
const dados = { nome: 'Márcio', email: 'marcio@gt.com.br' }; // Chave secreta para assinar o token
const chaveSecreta = 'minha_chave_secreta'; // Configurar a expiração para 10 minutos
const expiracao = Math.floor(Date.now() / 1000) + (10 * 60); // 10 minutos
// Função para gerar o token
const gerarToken = (dados) => { return jwt.sign(dados, chaveSecreta, { expiresIn: expiracao }); }; // Gerar o token
const token = gerarToken(dados); console.log('Token gerado:', token);
```

Execute o script para gerar o token:

bash


 Copiar código

```
node gerarToken.js
```

2. Verificar o Token

Vamos criar um arquivo `server.js` para verificar a validade do token.

javascript

 Copiar código

```
// server.js
const jwt = require('jsonwebtoken'); // Chave secreta para verificar o token
const chaveSecreta = 'minha_chave_secreta'; // Token gerado (substitua pelo token gerado no `gerarToken.js`)
const token = 'SEU_TOKEN_AQUI'; try { const decoded = jwt.verify(token, chaveSecreta); console.log('Token válido:', decoded); } catch (error) { console.error('Erro ao verificar token:', error.message); }
```

Execute o script para verificar o token:

```
bash
```

[Copiar código](#)

```
node server.js
```

Conclusão

Configurar a expiração dos tokens é fundamental para a segurança da sua aplicação. Certifique-se de definir um tempo de expiração adequado e tratar a renovação dos tokens quando necessário.

4.1. Vídeo Aula

dia 41 jwt vd4 converted

