Autenticação e Autorização

1. Conceitos de autenticação e autorização\*\*:

-Autenticação: O processo de verificar a identidade de um usuário ou sistema. É o ato de confirmar que alguém é quem diz ser. Exemplos comuns incluem login com nome de usuário e senha, uso de tokens, etc.

-Autorização: O processo de conceder permissão para acessar recursos específicos. É realizada após a autenticação e define o que um usuário autenticado pode ou não fazer. Exemplos incluem permissões de leitura, escrita, exclusão, etc.

Métodos de Autenticação em APIs

2. Métodos de autenticação comuns em APIs:

a. Autenticação baseada em senha:

- O usuário fornece um nome de usuário e senha que são verificados pelo servidor. Se as credenciais estiverem corretas, o acesso é concedido.

b. OAuth 2.0:

- Um protocolo de autorização que permite que aplicativos de terceiros obtenham acesso limitado a recursos de um usuário sem expor as credenciais do usuário. Utiliza tokens de acesso.

c. JWT (JSON Web Token):

- Um token compacto, autossuficiente e portátil usado para transmitir informações entre partes como um objeto JSON. É assinado digitalmente, o que garante sua integridade e autenticidade.

d. API Keys:

- Um token gerado pelo servidor que o cliente usa em suas solicitações para identificar e autenticar a aplicação que está fazendo a solicitação.

Autenticação com JSON Web Token (JWT)

3. Autenticação com JSON Web Token (JWT):

- JWT é um token baseado em JSON que é transmitido entre o cliente e o servidor para autenticação. Ele é composto por três partes: Header (cabeçalho), Payload (corpo) e Signature (assinatura).

- Header: Contém informações sobre o tipo de token e o algoritmo de assinatura.

- Payload: Contém as informações do usuário e outras reivindicações (claims).

- Signature: É usada para verificar se o token não foi alterado. É criada combinando o header e o payload com uma chave secreta usando um algoritmo específico.

Criptografia Simétrica e Assimétrica

4. Criptografia Simétrica e Assimétrica:

- Criptografia Simétrica: Utiliza a mesma chave para cifrar e decifrar os dados. É rápida e eficiente, mas a distribuição segura da chave é um desafio. Exemplo: AES (Advanced Encryption Standard).

- Criptografia Assimétrica: Utiliza um par de chaves – uma chave pública para cifrar e uma chave privada para decifrar. A chave pública pode ser distribuída livremente, mas a chave privada deve ser mantida em segredo. Exemplo: RSA.

Criptografia com o Módulo Crypto do Node.js

5. Criptografia com o módulo crypto do Node.js:

- O módulo `crypto` do Node.js fornece funcionalidades de criptografia. Você pode usá-lo para criar hashes, HMACs (Hash-based Message Authentication Code), cifras simétricas e assimétricas.

- Exemplo de uso para cifrar e decifrar dados:

```javascript

const crypto = require('crypto');

// Criptografar

const algorithm = 'aes-256-cbc';

const key = crypto.randomBytes(32);

const iv = crypto.randomBytes(16);

const cipher = crypto.createCipheriv(algorithm, key, iv);

let encrypted = cipher.update('Hello, World!', 'utf-8', 'hex');

encrypted += cipher.final('hex');

// Descriptografar

const decipher = crypto.createDecipheriv(algorithm, key, iv);

let decrypted = decipher.update(encrypted, 'hex', 'utf-8');

decrypted += decipher.final('utf-8');

console.log('Decrypted:', decrypted);

```

Pacotes Populares para Autenticação e Criptografia em Node.js

6. Pacotes populares para autenticação e criptografia em Node.js:

a. Nome do pacote: `jsonwebtoken`

- Principais funcionalidades: Criação, verificação e decodificação de tokens JWT.

- Como instalar e configurar:

```bash

npm install jsonwebtoken

```

```javascript

const jwt = require('jsonwebtoken');

const token = jwt.sign({ userId: 123 }, 'secretKey', { expiresIn: '1h' });

const decoded = jwt.verify(token, 'secretKey');

```

b. Nome do pacote: `bcrypt`

- Principais funcionalidades: Hashing de senhas, verificação de hashes.

- Como instalar e configurar:

```bash

npm install bcrypt

```

```javascript

const bcrypt = require('bcrypt');

const saltRounds = 10;

bcrypt.hash('password123', saltRounds, (err, hash) => {

// Store hash in your password DB.

bcrypt.compare('password123', hash, (err, result) => {

// result == true

});

});

```

c. Nome do pacote: `passport`

- Principais funcionalidades: Middleware de autenticação para Node.js, suporta várias estratégias de autenticação (e.g., local, OAuth, JWT).

- Como instalar e configurar:

```bash

npm install passport passport-local

```

```javascript

const passport = require('passport');

const LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy(

function(username, password, done) {

// Authenticate user here

}

));

app.post('/login', passport.authenticate('local', {

successRedirect: '/',

failureRedirect: '/login'

}));

```