

1. Autenticação com JWT no Node.js

Este capítulo apresenta alguns conceitos sobre como podemos realizar a autenticação e autorização com JWT em um projeto Node.js com Express.js.

1.1 Introdução

JSON Web Tokens (JWT) foram introduzidos como um método de comunicação segura entre duas partes. Ele foi introduzido com a especificação RFC 7519 pela Internet Engineering Task Force (IETF). Embora possamos usar o JWT com qualquer tipo de método de comunicação, hoje, o JWT é muito popular para lidar com autenticação e autorização via HTTP.

Primeiro, precisamos nos lembrar de algumas características do HTTP:

- O HTTP é um protocolo sem estado (*stateless protocol*), o que significa que uma requisição HTTP não mantém o estado entre outras requisições enviadas ao servidor. Assim, o servidor não tem conhecimento de nenhuma requisição anterior enviada pelo mesmo cliente.
- Desta forma, as requisições HTTP devem ser autocontidas. Elas devem incluir as informações sobre as requisições anteriores que o usuário fez na própria requisição.
- Existem algumas maneiras de fazer isso, no entanto, a forma mais comum é definir um ID de sessão, que é uma referência às informações do usuário. Desta forma, o servidor armazenará este ID de sessão na memória ou em um banco de dados. O cliente enviará cada requisição com este ID de sessão. O servidor pode então buscar informações sobre o cliente usando esta referência.

Veja abaixo um diagrama de como funciona a autenticação baseada em sessão (*session-based authentication*):



Normalmente, esse ID de sessão é enviado ao usuário como um cookie. Cabe lembrar que um cookie nada mais é que um pequeno arquivo de texto que contém uma etiqueta de identificação exclusiva, colocada no seu computador por um site. Neste arquivo, várias informações podem ser armazenadas, desde as páginas visitadas, manter informações de

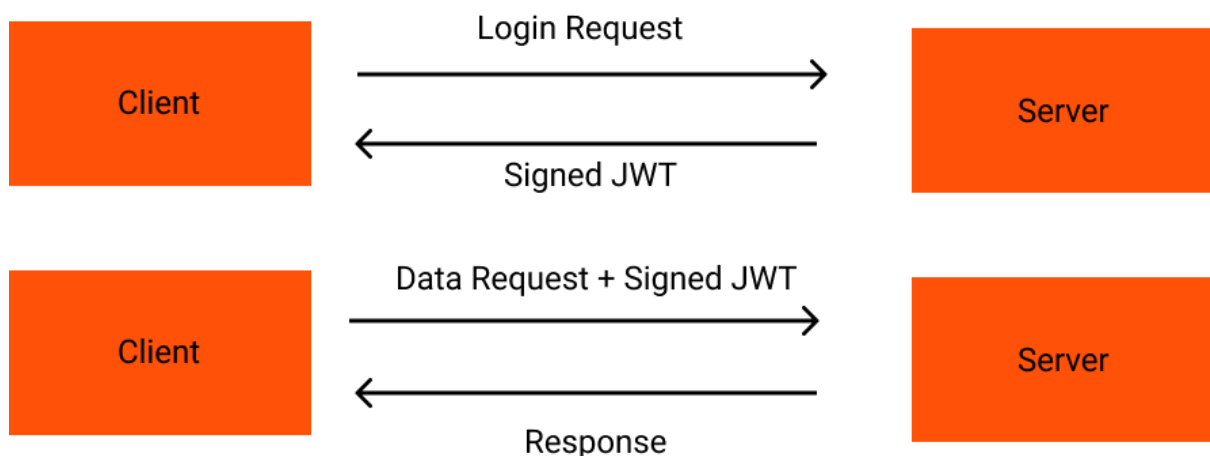
login, até os dados fornecidos voluntariamente ao site. Existem vários tipos de cookies, por exemplo:

- Cookies de sessão: Também chamado de cookie transitório, é apagado quando você fecha o navegador de internet. Ele é armazenado na memória temporária do computador e não é retido depois que o navegador é encerrado. Eles normalmente armazenam informações na forma de uma identificação que não coleta dados pessoais do usuário.
- Cookies persistentes: Também chamado de cookie permanente, é armazenado em seu disco rígido até expirar (cookies persistentes são definidos com datas de expiração) ou até você excluir. Os cookies persistentes são usados para coletar informações de identificação sobre o usuário, como comportamento de navegação na internet ou preferências para um site específico.
- Cookies maliciosos: Os cookies normalmente não comprometem a segurança, mas há uma tendência crescente de cookies maliciosos. Esses tipos de cookies podem ser usados para armazenar e acompanhar sua atividade online. Eles rastreiam você e seus hábitos de navegação ao longo do tempo, para construir um perfil de seus interesses.

Por outro lado, com o JWT, quando o cliente envia uma requisição de autenticação ao servidor, ele enviará um token JSON de volta ao cliente, que inclui todas as informações sobre o usuário com a resposta. Assim, o cliente enviará este token junto com todas as solicitações subsequentes. Assim, o servidor não terá que armazenar nenhuma informação sobre a sessão.

Entretanto, há um pequeno problema com essa abordagem. Qualquer pessoa pode enviar uma requisição falsa com um token JSON falso e fingir ser alguém que não é. Por exemplo, digamos que após a autenticação, o servidor envie de volta um objeto JSON com o nome de usuário e o tempo de expiração de volta para o cliente. Portanto, como o objeto JSON é legível, qualquer pessoa pode editar essas informações e enviar uma requisição. O problema é que não há como validar essa requisição.

Porém, é aqui que entra a assinatura do token. Portanto, em vez de apenas enviar de volta um token JSON simples, o servidor enviará um token assinado, que pode verificar se as informações não foram alteradas. Aqui está o diagrama de como o JWT funciona:



1.1.1 Estrutura de um JWT

Veja a estrutura de um JWT por meio de um token de exemplo:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MzkwMjQyLCJpcyI6IjE6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o

Como podemos ver na imagem, existem três seções deste JWT, cada uma separada por um ponto. A primeira seção do JWT é o cabeçalho, que é uma string codificada em Base64. A codificação Base64 é uma forma de garantir que os dados não estão corrompidos, Ela não os compacta ou criptografa, mas simplesmente os codifica de uma maneira que a maioria dos sistemas possa entender. Dessa forma pode ler qualquer texto codificado em Base64 simplesmente decodificando-o. Se você decodificou o cabeçalho, seria algo semelhante a isto:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Como podemos ver acima, a seção do cabeçalho contém o algoritmo de hash, que foi usado para gerar a assinatura e o tipo do token.

A segunda seção contém os dados (*payload*), o objeto JSON, que foi enviado de volta ao usuário. Uma vez que é codificado apenas em Base64, pode ser facilmente decodificado por qualquer pessoa. Recomenda-se não incluir quaisquer dados confidenciais nos JWTs, como senhas ou informações de identificação pessoal. Normalmente, o corpo do JWT será parecido com isto:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

Na maioria das vezes, a propriedade **sub** conterá o ID do usuário, a propriedade **iat**, que é a abreviação de issued at (emitido em), é a data/hora de quando o token é emitido. Você também pode ver algumas propriedades comuns, como **exp** ou **exp**, que é o tempo de expiração do token.

A seção final é a assinatura do token. Este segredo (**secret**) é uma string aleatória que apenas o servidor deve saber. Ele é gerado por hash, sabendo que nenhum hash pode ser convertido de volta para o texto original e mesmo uma pequena mudança na string original resultará em um hash diferente. Portanto, o segredo não pode ser submetido a engenharia reversa. Quando essa assinatura é enviada de volta ao servidor, ela pode verificar se o cliente não alterou nenhum detalhe no objeto.

De acordo com os padrões, o cliente deve enviar este token para o servidor através da requisição HTTP em um cabeçalho denominado **Authorization** (autorização) com o formato chamado Bearer [JWT_TOKEN]. Portanto, o valor do cabeçalho de autorização será semelhante a:

```
Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o
```

1.2 Exemplo de JWT com Express

Neste tutorial, criaremos um aplicativo para gerenciar os cursos do IFRS utilizando dois serviços. Um serviço será responsável pela autenticação do usuário e o outro será responsável pelo gerenciamento dos cursos. Haverá dois tipos de usuários - administradores e professores. Os administradores poderão visualizar e adicionar novos cursos, enquanto os professores só poderão visualizá-los.

Para começar, crie um diretório chamado node-jwt. Em seguida, vamos iniciar um projeto Node com auxílio do gerenciador de pacotes npm. Assim, digite os seguintes comandos no terminal:

```
cd node-jwt
npm init -y
```

Este comando cria um arquivo chamado package.json. Cabe lembrar que se você especificar a opção -y no comando, o npm aceita automaticamente os valores padrões.

Agora que já temos o nosso projeto criado, vamos começar a criar nosso servidor web. Assim, vamos abrir o projeto no Visual Studio Code, digitando o seguinte comando no terminal, dentro do diretório do nosso projeto:

```
code .
```

Em seguida, vamos realizar o download e instalação de nossas dependências do projeto. Neste caso, vamos utilizar o Express. Assim digite o seguinte comando no terminal, dentro do diretório do projeto:

```
npm i express
```

OBS: Você pode usar o terminal que fica integrado ao Visual Studio Code

Agora, vamos instalar o Nodemon como dependência de desenvolvimento no projeto. O Nodemon é um utilitário que monitora quaisquer mudanças em seu código fonte e automaticamente reinicia seu servidor.

```
npm i --save-dev nodemon
```

Pronto!

1.2.1 Configure o servidor

Primeiro, vamos criar um arquivo chamado server.js com o seguinte código:

```
const express = require('express');
const app = express();
app.listen(3000, () => {
  console.log('Serviço iniciado na porta 3000');
});
```

Em seguida, vamos alterar o arquivo package.json para usarmos o nodemon:

```
{
  "name": "jwt",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "dev": "nodemon",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

Assim, digite o seguinte comando no terminal:

```
npm run dev
```

Observe que ele executou o script “dev” que acabamos de criar e executou o nodemon.

```
PS C:\Users\Mauri\Downloads\node-jwt> npm run dev
> jwt@1.0.0 dev
> nodemon

[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Serviço iniciado na porta 3000
```

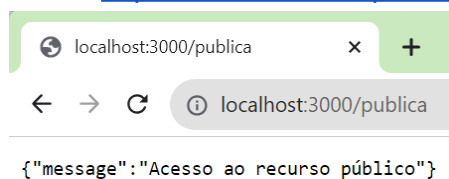
1.2.2 Rotas públicas e privadas

Em nosso teste, precisaremos criar uma rota que seja pública, onde todos tenham acesso, e uma privada, somente usuários autorizados. Assim, altere o arquivo server.js:

```
...
// Rota publica
app.get('/publica', (req, res) => {
  res.json({ message: 'Acesso ao recurso público' });
})
// Rota privada
app.get('/privada', (req, res) => {
  res.json({ message: 'Acesso ao recurso protegido permitido' });
})
...
```

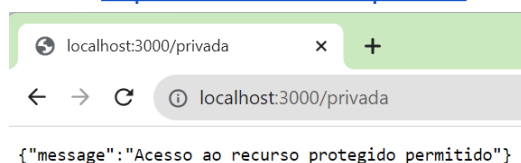
Faça um rápido teste usando o seu navegador informando as duas rotas:

- <http://localhost:3000/publica>



A screenshot of a web browser window. The address bar shows 'localhost:3000/publica'. Below the address bar, the response is displayed as a JSON object: {"message": "Acesso ao recurso público"}.

- <http://localhost:3000/privada>



A screenshot of a web browser window. The address bar shows 'localhost:3000/privada'. Below the address bar, the response is displayed as a JSON object: {"message": "Acesso ao recurso protegido permitido"}.

Observe que, no momento, conseguimos acessar tanto a rota pública quanto a privada.

1.2.3 Variáveis de ambiente

As variáveis de ambiente guardam os dados de configuração do sistema. Elas geralmente armazenam informações sensíveis como, por exemplo, chaves de APIs, credenciais de acesso a bancos de dados, localização de arquivos, portas HTTPs, etc. Já que as variáveis de ambiente contêm informações privadas, é importante que elas não fiquem expostas. Sendo assim, uma boa prática é mantê-las separadas do código.

O [dotenv](#) é uma biblioteca que serve justamente para gerenciar as variáveis de ambiente dentro de um projeto Node.js. Ela armazena a configuração dessas variáveis em um ambiente separado do código da aplicação. Resumidamente, a biblioteca dotenv faz o seguinte:

1. Carrega as variáveis de ambiente do arquivo .env para o processo do Node.js.
2. Permite que você acesse essas variáveis de ambiente em seu código através do objeto process.env.
3. Torna mais fácil a gestão de variáveis de ambiente em diferentes ambientes de desenvolvimento.
4. Ajuda a manter informações sensíveis, como chaves de API ou senhas, separadas do código-fonte, aumentando a segurança.

Assim, digite o seguinte comando no terminal para instalar essa biblioteca:

```
npm i --save-dev dotenv
```

OBS: A biblioteca dotenv é uma forma popular de carregar a configuração do seu aplicativo Node.js a partir de um arquivo .env em um ambiente de desenvolvimento. Você não deve precisar disso no ambiente de produção pois a maioria dos provedores de hospedagem (por exemplo, Digital Ocean, Heroku, AWS) fornece seu próprio mecanismo para você injetar variáveis de ambiente com segurança no ambiente em que seu aplicativo está sendo executado.

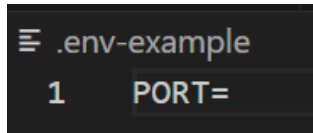
Com o pacote [dotenv](#) instalado em seu projeto, o primeiro passo para utilizá-lo é criar um arquivo chamado “.env” dentro do diretório raiz da sua aplicação. Esse arquivo é responsável por guardar as variáveis de ambiente do seu projeto, escritas no formato NOME=valor. Por exemplo, você pode definir uma variável para a porta com o valor 3000, assim: PORT=3000.

OBS: É muito importante que você não envie o arquivo .env para o repositório do Github que você estiver utilizando, já que ele pode conter dados confidenciais, como chaves de autenticação e senhas. Adicione o arquivo ao .gitignore para evitar de enviá-lo a um repositório público por acidente.

Mas, se você não pode fazer commit do arquivo .env, é necessário alguma outra forma para que os demais desenvolvedores do projeto saibam quais variáveis de ambiente são necessárias para executar o software. É comum que os desenvolvedores listem as variáveis de ambiente necessárias para executar o programa em um README ou documentação interna semelhante. Outra prática muito comum é criar um arquivo de exemplo do .env que

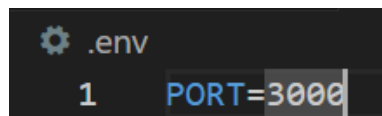
contenha o nome de todas as variáveis de ambiente e enviar esse arquivo para o Github. Dessa forma, outras pessoas que contribuirão com o desenvolvimento do projeto podem facilmente preencher o seu próprio .env com as informações do seu próprio ambiente, sem que nenhuma variável seja esquecida.

Assim, vamos criar um arquivo .env-example na raiz do nosso projeto:



```
1 PORT=
```

E, em seguida criaremos nosso arquivo chamado “.env” na raiz do projeto que conterá todas as nossas variáveis de ambiente, no momento fica assim:



```
1 PORT=3000
```

Como em qualquer outro módulo do Node.js, é necessário importar o módulo para conseguir usar suas funcionalidades. Assim, altere o arquivo server.js:

```
const express = require("express");
require("dotenv").config();
...
```

Agora que importamos o módulo, podemos utilizá-lo dentro da nossa aplicação. A próxima etapa é acessar nossas variáveis de ambiente. Fazemos isso através do process.env. O formato usado para acessar as variáveis de ambiente é process.env.NOME_VARIAVEL.

Altere o arquivo server.js para lermos esta variável de ambiente:

```
const express = require("express");
require("dotenv").config();

const app = express();
const port = process.env.PORT || 3000;

// Rota publica
app.get("/publica", (req, res) => {
  res.json({ message: "Acesso ao recurso público" });
});

// Rota privada
app.get("/privada", (req, res) => {
  res.json({ message: "Acesso ao recurso protegido permitido" });
});
app.listen(port, () => {
```



```
console.log(`Serviço iniciado na porta ${port}`);  
});
```

Inicie o servidor com o seguinte comando no terminal:

```
npm run dev
```

Funcionou!

1.2.3 Autenticação

Autenticação é o processo de verificação das credenciais que um usuário fornece com aquelas armazenadas em um sistema para provar que o usuário é quem diz ser. Se as credenciais corresponderem, você concederá acesso. Se não, você nega.

Assim, vamos começar a pensar em como fazer a autenticação deste serviço. Primeiro, vamos instalar o módulo [jsonwebtoken](#), que é usado para gerar e verificar tokens JWT.

```
npm i jsonwebtoken
```

Agora, vamos adicionar esse módulo em nosso aplicativo. Altere server.js:

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

Agora podemos criar um segredo para lidar com a requisição de login do usuário. Altere o arquivo .env:

```
PORT=3000  
SECRET=suaChaveSecretaSecreta
```

E o arquivo .env-example

```
PORT=  
SECRET=
```

Este é o seu segredo para assinar o token JWT. Você nunca deve compartilhar esse segredo, caso contrário, um malfeitor poderia usá-lo para forjar tokens JWT para obter acesso não autorizado ao seu serviço. Quanto mais complexo for esse token de acesso, mais seguro será o seu aplicativo. Portanto, tente usar uma string aleatória complexa para este token, mas, para este exemplo, este segredo serve.

Idealmente, devemos usar um banco de dados para armazenar informações dos usuários. Mas para manter esse exemplo simples, vamos deixar no código um username e password válidos. Logo, vamos criar uma nova rota /login que vai receber um usuário e senha e, caso esteja ok, retornará um JWT para o cliente:

```
// Rota para fazer login e gerar um token JWT
app.post("/login", (req, res) => {
  const { username, password } = req.body;
  // Verifica se o usuário e senha existem em nossa "base de dados"
  if (username === "usuario" && password === "senha") {
    const id = 1; //esse id viria do banco de dados
    const token = jwt.sign({ id }, process.env.SECRET, {
      expiresIn: 300, // expira em 5min
    });
    res.json({ token });
  } else {
    res.status(401).json({ message: "Credenciais inválidas" });
  }
});
```

Neste exemplo, o cliente posta na URL /login um username e um password, que simulo uma ida ao banco. Estando ok, o banco me retornaria o ID deste usuário, que simulei com uma constante. Esse ID está sendo usado como payload do JWT que está sendo assinado, mas poderia ter mais informações conforme a sua necessidade. Além do payload, é passado o SECRET, que está armazenado em uma variável de ambiente. Por fim, adicionei uma expiração de 5 minutos para este token. Caso o user e password não coincidam, será devolvido um erro ao usuário.

Veja que precisamos que os dados venham no formato JSON. Para analisar o conteúdo do corpo (body) da requisição, precisamos usar o body-parser, um módulo do Node.js capaz de converter o body da requisição para vários formatos, dentre eles, o JSON.

```
const express = require("express");
const jwt = require("jsonwebtoken");
require("dotenv").config();

const app = express();

// Configura o parser para requisições com JSON
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
...
```

Nossa rota de autenticação está pronta.

Inicie o servidor com o seguinte comando no terminal:

```
npm run dev
```

1.2.3 Testando nossa rota de autenticação

Vimos anteriormente que o Visual Studio Code oferece uma extensão chamada [REST Client](#) que possibilita que você crie scripts com as configurações de requisições HTTP e os execute sem grandes complicações.

O uso da REST Client é bastante simples, mas, primeiro, busque pela extensão REST Client e instale-a. Agora, em seu projeto, crie uma pasta com o nome “request”. Em seguida, dentro desta pasta, crie um arquivo chamado `login.http`. No arquivo `login.http` vamos configurar a requisição para o método HTTP POST. A URL é <http://localhost:3000/login/> e temos que informar os campos que vamos enviar para a API:

```
@baseUrl = http://localhost:3000
### Autenticação
POST {{baseUrl}}/login/
Content-Type: application/json

{
  "username": "usuario",
  "password": "senha"
}
```

Você deve obter o token de acesso como resposta:

```
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 161
5 ETag: W/"a1-HjV+jyL6ol+UAH8p1ItP1IoZDtY"
6 Date: Fri, 27 Oct 2023 18:46:59 GMT
7 Connection: close
8
9 {
10   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1bmFtZSI6ImVvaG4iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOiJlZ20tTg0MzI0MTI5LjYyVzVWYyHjBxvHr3zqCtkXfY-Nz1y8sRj2TMD9wyeZY"
11 }
```

Funcionou! Neste exemplo, temos uma rota para fazer login, que gera um token JWT após a autenticação. Tente informar um usuário e senha incorretos.

```
@baseUrl = http://localhost:3000
```

```
### Autenticação
POST {{baseUrl}}/login/
Content-Type: application/json

{
  "username": "usuario_errado",
  "password": "senha"
}
```

E veja a resposta:

```
Response(7ms) X
1 HTTP/1.1 401 Unauthorized
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 36
5 ETag: W/"24-Yaa43+dAxmjUdGy6wp9zU4ARpmw"
6 Date: Sat, 28 Oct 2023 00:06:55 GMT
7 Connection: close
8
9 {
10   "message": "Credenciais inválidas"
11 }
```

1.2.4 Autorização

Autorização é o processo de verificar se você tem permissão para acessar uma área de uma aplicação ou executar ações específicas, com base em determinados critérios e condições estabelecidos pela aplicação. Assim, vamos criar uma função [middleware](#) para que, dada uma requisição que está chegando, seja verificada se ela possui um JWT válido.

```
function verifyJWT(req, res, next) {
  const token = req.headers.authorization.split(' ')[1];
  if (!token) return res.status(401).json({ message: "Token não fornecido" });

  jwt.verify(token, process.env.SECRET, function (err, decoded) {
    if (err)
      return res.status(500).json({ auth: false, message: "Token inválido" });

    // se tudo estiver ok, salva no request para uso posterior
    req.userId = decoded.id;
    next();
  });
}
```

Os [middlewares](#) são funções que podem tratar os inputs e outputs das rotas antes e/ou depois que uma rota é processada, ou seja, você pode criar um middleware que intercepta e verifica se usuário está autenticado ou qualquer outra coisa que precise ser feita antes de devolver uma resposta para a requisição.

Assim, os middlewares são funções que têm acesso ao objeto de solicitação (`req`), o objeto de resposta (`res`), e a próxima função de middleware no ciclo requisição-resposta do aplicativo. A próxima função middleware é comumente denotada por uma variável chamada `next`.

Neste middleware, obtemos o token a partir do cabeçalho (headers) “authorization”. Em seguida, verificamos a existência do token. Caso exista, verificamos a autenticidade desse token usando a função `verify()`, usando a variável de ambiente com o SECRET. Caso ele não consiga verificar o token, irá gerar um erro. Em seguida chamamos a função `next` que passa para o próximo estágio de execução das funções no pipeline do middleware do Express, mas não antes de salvar a informação do ID do usuário para a requisição, visando poder ser utilizado pelo próximo estágio.

OBS: Como o cabeçalho de autorização tem um valor no formato Bearer [JWT_TOKEN], dividimos o valor pelo espaço e separamos o token. Lembre-se do formato Bearer [JWT_TOKEN] como sendo:

```
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjMONTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHMI6arZ3Y922BhjWgQzWXcXNrZ0ogtVhfEd2o
```

Por esse motivo, fizemos o seguinte código para pegar o token:

```
const token = req.headers.authorization.split(' ')[1];
```

Para usarmos esse middleware, basta inserirmos sua referência na chamada GET /privada que já existia em nossa API:

```
...
// Rota privada
app.get('/privada', verifyJWT, (req, res, next) => {
  res.json({ message: 'Acesso ao recurso protegido permitido' });
})
...
```

Assim, antes de responder os GETs da rota “privada”, a API vai criar essa camada intermediária de autorização baseada em JWT, que obviamente vai bloquear requisições que não estejam autenticadas e autorizadas, conforme as suas regras para tal.

Vamos testar... no arquivo login.http insira a seguinte chamada para a API:

```
### Rota Privada
GET {{baseUrl}}/privada/
```

E veja o resultado:

```
Response(26ms) X
1 HTTP/1.1 401 Unauthorized
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 34
5 ETag: W/"22-NOeD41Az20qV9VmfwPD0Zq2zKE"
6 Date: Sat, 28 Oct 2023 00:59:11 GMT
7 Connection: close
8
9 {
10   "message": "Token não fornecido"
11 }
```

Veja que recebemos como resposta que o token JWT não foi fornecido. Sim, precisamos nos logar antes e usar este token na requisição da rota privada.

Faça o login:

```
### Autenticação
POST {{baseUrl}}/login/
Content-Type: application/json

{
  "username": "usuario",
  "password": "senha"
}
```

E salve o token:

```
Response(5ms) X
Response(5ms)
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 149
5 ETag: W/"95-RN416ce/1U6UBCda3F/r3pBMP80"
6 Date: Sat, 28 Oct 2023 01:41:40 GMT
7 Connection: close
8
9 {
10   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjY4NDUzMzAwLCJleHAiOjE2OTg0NTc2MDB9.3tPNKpt_PYBvTkrdPSRmRGU9p_vBPuu8PhKuq0pi4VI"
11 }
```

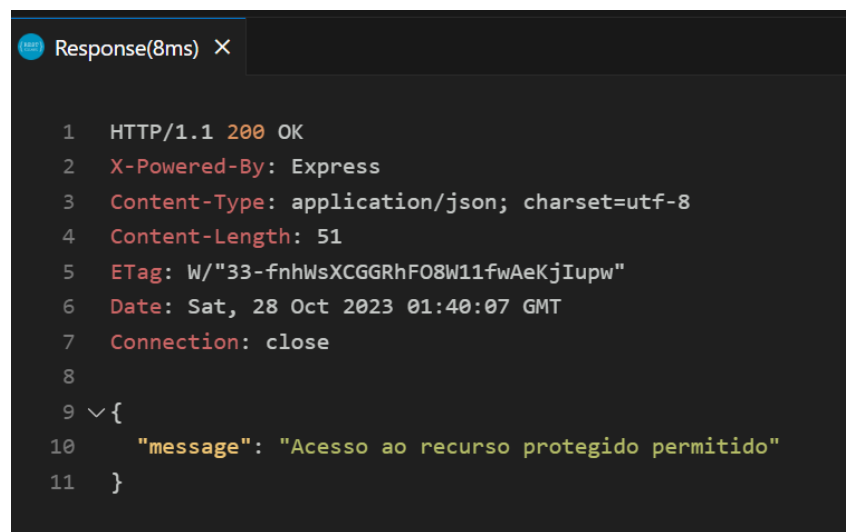
Altere o arquivo login.http para usarmos esse token:

```
@authToken
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNjk4NDUzMzAwLCJleHAiOjE2OTg0NTc2MDB9.3tPNKpt_PyBvTkrdPSRmRGU9p_vBPuu8PhKuq0pi4VI
```

E, agora, vamos testar nossa chamada para a rota privada em login.http:

```
### Rota Privada
GET {{baseUrl}}/privada/
Authorization: Bearer {{authToken}}
```

Veja o resultado:



```
Response(8ms) X
1 HTTP/1.1 200 OK
2 X-Powered-By: Express
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 51
5 ETag: W/"33-fnhWsXCGGRhFO8W11fwAeKjIupw"
6 Date: Sat, 28 Oct 2023 01:40:07 GMT
7 Connection: close
8
9 {
10   "message": "Acesso ao recurso protegido permitido"
11 }
```

Funcionou!!!

Referências

- Site Oficial do Express. Disponível em: <http://expressjs.com/pt-br/guide/routing.html>
- Site MDN Web Docs. Introdução ao Express/Node. Disponível em: https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introdu%C3%A7%C3%A3o
- Página Luiz Duarte. Autenticação JSON Web Token (JWT) em Node.js. Disponível em: <https://www.luiztools.com.br/post/autenticacao-json-web-token-jwt-em-nodejs/>
- Site MDN Web Docs. Códigos de status de respostas HTTP . Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>