

**Objetivos:**

- I. Introdução;
- II. Autenticação usando algoritmos de hash;
- III. Autenticação com tokens;
- IV. Middleware.

**I. Introdução**

**Autenticação** é o processo de verificar a identidade de um usuário ou cliente. Ele confirma se quem está tentando acessar o sistema é realmente quem diz ser. Um exemplo comum é o login com nome de usuário e senha.

**Autorização** define os recursos ou funcionalidades que um usuário autenticado pode acessar dentro do sistema.

Importância da autenticação e autorização:

- Segurança: protege os dados dos usuários e da aplicação contra acessos não autorizados;
- Privacidade: garante que apenas usuários autorizados tenham acesso a informações confidenciais;
- Integridade: evita modificações não autorizadas nos dados do sistema;
- Disponibilidade: controla o acesso aos recursos do sistema, prevenindo sobrecargas e acessos indevidos.

Analogia:

- Autenticação é como um passaporte: ele prova quem você é.
- Autorização é como um visto: define os locais que você tem permissão para visitar.

**II. Autenticação usando algoritmos de hash**

Na autenticação baseada em senhas, é fundamental armazenar as senhas de forma segura no servidor. Para isso, utiliza-se o processo de hashing, que converte uma senha em um valor único e mascarado. Esse valor não pode ser revertido para a senha original.

**Hashing** é o processo de transformar um dado (como uma senha) em uma sequência fixa de caracteres. Por exemplo, o texto "abc" pode ser transformado no hash:

"\$argon2id\$v=19\$m=65536,t=3,p=4\$NdJY6jEmSrMg+hVDXSHn7A\$pNRYx999tah9r21+QKzrlrMd24+c2+48ZrtBWB  
EDrUw"

Exemplos de algoritmos de hash:

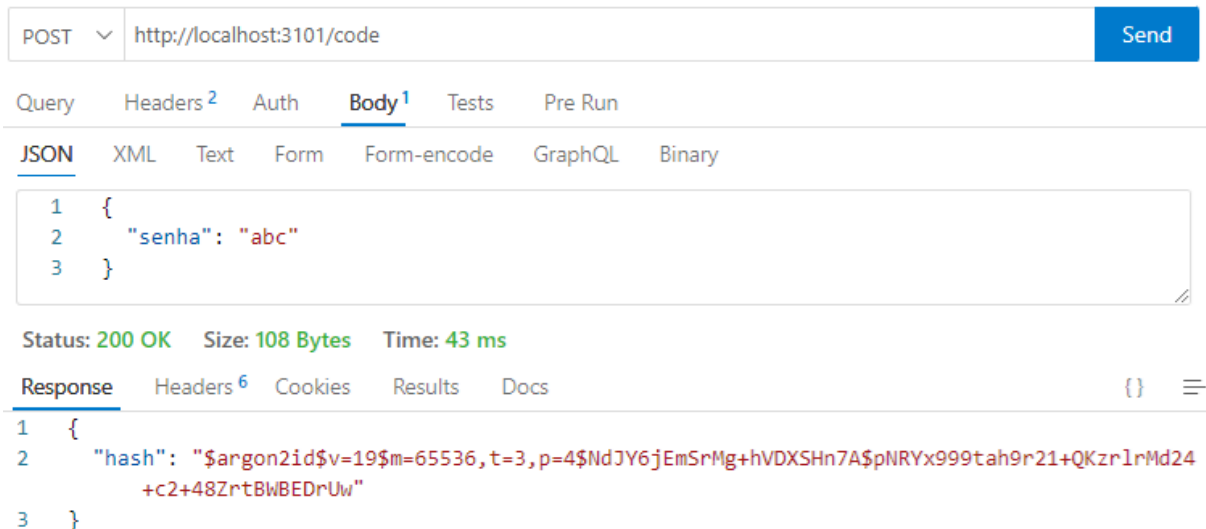
- Bcrypt: algoritmo amplamente usado e seguro para hashing de senhas (<https://www.npmjs.com/package/bcrypt>);
- argon2: recomendado como o mais seguro e vencedor da Password Hashing Competition de 2015 (<https://www.npmjs.com/package/argon2>);

- SHA-256: menos seguro para hashing de senhas devido à sua velocidade, mas ainda usado em contextos específicos.

Instalação do pacote: `npm i argon2`

Descrição do processo de codificação e verificação implementado no código a seguir:

1. O cliente envia uma senha para o servidor através do endpoint `/code`. O servidor retorna o hash da senha gerado utilizando Argon2;



POST `http://localhost:3101/code` Send

Query Headers <sup>2</sup> Auth Body <sup>1</sup> Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

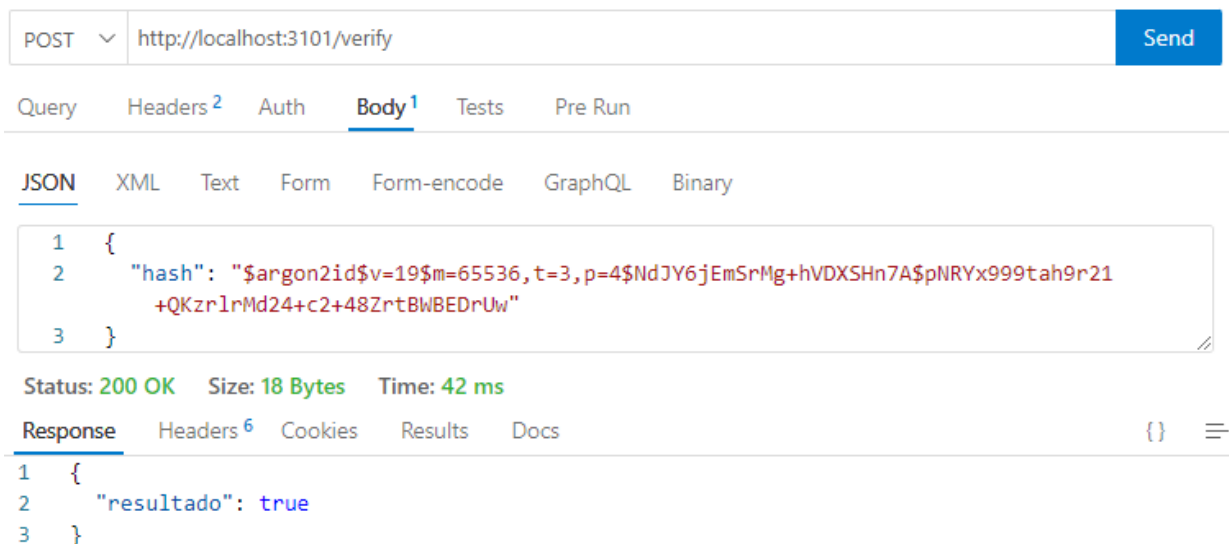
```
1 {
2   "senha": "abc"
3 }
```

Status: 200 OK Size: 108 Bytes Time: 43 ms

Response Headers <sup>6</sup> Cookies Results Docs {} ≡

```
1 {
2   "hash": "$argon2id$v=19$m=65536,t=3,p=4$NdJY6jEmSrMg+hVDXSHn7A$pNRYx999tah9r21+QKzr1rMd24+c2+48ZrtBwBEDrUw"
3 }
```

2. O cliente pode enviar o hash para o endpoint `/verify` para verificar se a senha original corresponde ao hash armazenado no servidor. Neste exemplo, o servidor usa a senha "abc" como base de comparação, mas em um cenário real, a senha deveria estar armazenada de forma segura em um BD.



POST `http://localhost:3101/verify` Send

Query Headers <sup>2</sup> Auth Body <sup>1</sup> Tests Pre Run

JSON XML Text Form Form-encode GraphQL Binary

```
1 {
2   "hash": "$argon2id$v=19$m=65536,t=3,p=4$NdJY6jEmSrMg+hVDXSHn7A$pNRYx999tah9r21+QKzr1rMd24+c2+48ZrtBwBEDrUw"
3 }
```

Status: 200 OK Size: 18 Bytes Time: 42 ms

Response Headers <sup>6</sup> Cookies Results Docs {} ≡

```
1 {
2   "resultado": true
3 }
```

```
import express, { Request, Response } from "express";
import dotenv from "dotenv";
import argon2 from "argon2";
```

```
dotenv.config();

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

app.use(express.json());

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});

app.post("/code", async (req: Request, res: Response) => {
  const { senha } = req.body;
  const hash = await hashPassword(senha);
  res.json({ hash });
});

app.post("/verify", async (req: Request, res: Response) => {
  const { hash } = req.body;
  const senhaServidor = "abc"; // senha armazenada no servidor
  const resultado = await verifyPassword(senhaServidor, hash);
  res.json({ resultado });
});

async function hashPassword(password: string) {
  // codifica a senha
  const hash = await argon2.hash(password);
  return hash;
}

async function verifyPassword(password: string, hashedPassword: string) {
  // compara o hash com a senha
  return await argon2.verify(hashedPassword, password);
}
```

Observações importantes:

- O hash gerado nunca deve ser armazenado diretamente no código. Em um sistema real, ele deve ser salvo em um BD.
- Não use a senha original no código. No exemplo, "abc" foi usada apenas para fins de ilustração. Em um sistema seguro, as senhas são fornecidas pelos usuários e armazenadas de forma codificado em um BD.

### III. Autenticação com tokens

A autenticação com tokens é um método que utiliza pequenos pacotes de dados (tokens) para autenticar usuários em um sistema. Esses tokens são gerados pelo servidor após o usuário se autenticar com sucesso e são utilizados para validar futuras requisições sem a necessidade de reenviar as credenciais.

Analogia: imagine que você entra em um cinema e recebe um ingresso. Esse ingresso é como um token: ele prova que você pagou pela entrada e permite que você acesse a sala de cinema sem precisar mostrar novamente o comprovante de pagamento.

O JWT (JSON Web Token) é um padrão amplamente utilizado para autenticação. Ele é compacto, seguro e pode ser usado em diversos cenários, como APIs REST.

Um JWT é composto por três partes principais, separadas por pontos (.):

1. Header (cabeçalho): contém informações sobre o tipo do token (JWT) e o algoritmo de assinatura utilizado (como HMAC ou RSA);

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

2. Payload (carga útil): armazena as informações do usuário (claims) como um objeto JSON;

```
{
  "mail": "root@test.com",
  "role": "admin",
  "iat": 1736523180,
  "exp": 1736526780
}
```

3. Signature (assinatura): garantia de que o token não foi alterado. É gerada combinando o header, payload e uma chave secreta. Exemplo de token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtYWlsIjoicm9vdEB0ZXN0LmNvbSIzInJvbGU0IjZG1pb  
iIsIm1hdCI6MTczNjUyMzQzMiwiaXhwIjozNzY2NTI3MDMyfQ.mKTmhEM1EcGRry3rDAPB0xrT-  
8KhxI4eRQDDiS7HiBk
```

Fluxo de autenticação com JWT:

1. O usuário envia suas credenciais (ex.: e-mail e senha) ao servidor;
2. O servidor valida as credenciais;
3. Se as credenciais forem válidas, o servidor gera um token JWT e o envia ao cliente. A seguir tem-se o resultado dos passos 1 a 3:

The screenshot shows a REST client interface. The top bar indicates a POST request to `http://localhost:3101/login` with a 'Send' button. Below the bar, tabs for 'Query', 'Headers', 'Auth', 'Body', 'Tests', and 'Pre Run' are visible. The 'Body' tab is selected, showing a JSON payload: 

```
{
  "mail": "root@test.com",
  "password": "123456"
}
```

 Below the body, the status is '200 OK', size is '191 Bytes', and time is '4 ms'. The 'Response' tab is selected, showing a JSON response: 

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtYWlsIjoicm9vdEB0ZXN0LmNvbSI6InR5bGU0Ij01ZG1pbiIsIm1hdCI6MTczNjUyMzQzMiwiaXhwIjoxNzY2MjNTI3MDMyfQ.mKtMhEM1EcGRry3rDpB0xrT-8KhxI4eRQDDiS7HiBk"
}
```

4. O cliente utiliza o token em todas as requisições subsequentes, geralmente no cabeçalho Authorization com o prefixo Bearer;
5. O servidor valida o token antes de processar a requisição. A seguir tem-se o resultado dos passos 4 e 5:

POST  Send

Query Headers **Auth** Body Tests Pre Run

None Basic **Bearer** OAuth 2 NTLM AWS

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtYWlsIjoicm9vdEB0ZXN0LmNvbSIsInJvbGUiOiJhZG1pbilzImIhdCI6MTczNjUyMzQzMiwiaXhwIjojNzY2MjNTI3MDMyfQ.mKTmhEM1EcGRy3rDapB0xrT-8Khl4eRQDDiS7HiBk ← Sem aspas

Status: 200 OK Size: 73 Bytes Time: 5 ms

Response Headers Cookies Results Docs

```

1 {
2   "mail": "root@test.com",
3   "role": "admin",
4   "iat": 1736523432,
5   "exp": 1736527032
6 }
  
```

### Instalação do pacote e definição de tipos:

```
npm i jsonwebtoken
npm i @types/jsonwebtoken -D
```

Para testar, adicione a variável `JWT_SECRET` no arquivo `.env`. A variável `JWT_SECRET` pode ter qualquer senha, aqui foi utilizada `@123`:

```
PORT = 3101
JWT_SECRET = @123
```

```
import express, { Request, Response } from "express";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";
```

```
dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;
const SECRET_KEY = process.env.JWT_SECRET || "chave";

app.use(express.json());

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});

// Rota de login para gerar o token
app.post("/login", (req: Request, res: Response) => {
  const { mail, password } = req.body;

  // Validação simplificada (substitua por uma verificação em banco de dados)
  if (mail === "root@test.com" && password === "123456") {
    // Gera o token com as informações do usuário
    const token = jwt.sign({ mail, role: "admin" }, SECRET_KEY, {
      expiresIn: "1h",
    });
    res.json({ token });
  } else {
    res.status(401).json({ error: "Credenciais inválidas." });
  }
});

// Rota para checar token
app.post("/verify", (req: Request, res: Response) => {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    res.status(401).json({ error: "Token não fornecido." });
  } else {
    const token = authHeader.split(" ")[1];

    try {
      const decoded = jwt.verify(token, SECRET_KEY);
      res.json(decoded);
    } catch (err) {
      res.status(403).json({ error: "Token inválido ou expirado." });
    }
  }
});
```

Benefícios do JWT:

- Independência do servidor: não é necessário armazenar sessões no servidor;

- Facilidade de integração: amplamente suportado em diferentes plataformas e linguagens;
- Segurança: assinaturas digitais garantem a integridade do token.

Bearer token: é um token de segurança que garante que qualquer parte que o possua (um "portador", em inglês *bearer*) seja reconhecida pela outra parte como autorizada. O uso de um *Bearer token* não exige que o portador comprove a posse de material de chave criptográfica (conhecido como *proof-of-possession*, ou "prova de posse").

#### IV. Middleware

No contexto do Express, um middleware é uma função que tem acesso ao objeto de requisição (req), ao objeto de resposta (res) e à próxima função (next) no ciclo de requisição-resposta. Ele pode realizar as seguintes ações:

- Executar qualquer lógica necessária para processar a requisição;
- Modificar os objetos de requisição ou resposta;
- Encerrar o ciclo de requisição enviando uma resposta ao cliente;
- Chamar o próximo middleware ou controlador, usando a função next().

Fluxo de execução de uma requisição no código a seguir:

1. O cliente faz uma requisição POST para a rota /teste com um corpo JSON;
2. A requisição passa pelo middleware **intermediária**, que faz uma validação:
  - Se nome !== undefined, o middleware chama **next()**, permitindo que a execução continue para a função **objetivo**;
  - Caso contrário, o middleware envia uma resposta ao cliente com o código de status 401 e não permite que o fluxo continue, ou seja, a função **objetivo** não será executada.

A função objetivo foi chamada:

POST http://localhost:3101/teste

Query

Headers<sup>2</sup>

Auth<sup>1</sup>

**Body<sup>1</sup>**

Tests

1 {

2 "nome": "root@test.com"

3 }

Status: 200 OK

Size: 32 Bytes

Time: 15 ms

**Response**

Headers<sup>6</sup>

Cookies

Results

1 {

2 "message": "Olá root@test.com"

3 }

A função intermediária encerra a requisição:

POST	▼	http://localhost:3101/teste		
Query	Headers <sup>2</sup>	Auth <sup>1</sup>	<u>Body</u>	
<div>1</div>				
Status: 401 Unauthorized		Size: 27 Bytes		
<u>Response</u>	Headers <sup>6</sup>	Cookies	Result	
1	{			
2	"error": "Não autorizado"			
3	}			

```
import express, { NextFunction, Request, Response } from "express";
import dotenv from "dotenv";

dotenv.config();

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

app.use(express.json());

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});

function objetivo(req: Request, res: Response) {
  const { nome } = req.body;
  res.send({ message: `Olá ${nome}` });
}

function intermediaria(req: Request, res: Response, next: NextFunction): void {
  const { nome } = req.body;
  if (nome !== undefined) {
    next();
  } else {
    res.status(401).send({ error: "Não autorizado" });
  }
}

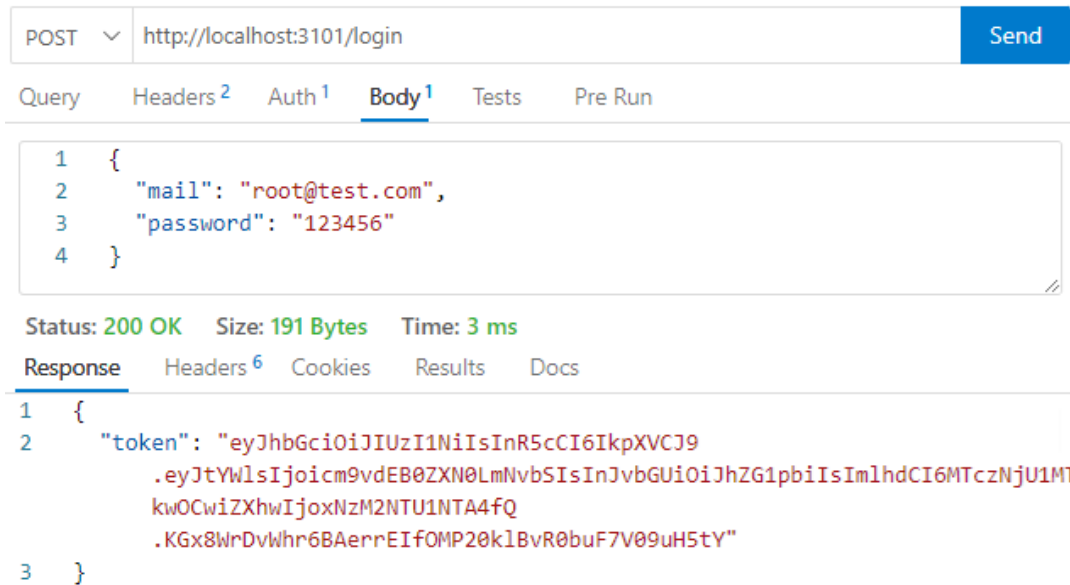
app.post("/teste", intermediaria, objetivo);
```

Em sistemas que utilizam o JWT para autenticação, o middleware desempenha um papel crucial: verificar a validade do token em cada requisição antes que a lógica da aplicação seja executada. Isso garante que apenas usuários autenticados possam acessar rotas protegidas.

Funcionamento do código a seguir de um middleware de autenticação usando JWT:

- O cliente envia os dados de login e o servidor retorna um token;





POST http://localhost:3101/login Send

Query Headers<sup>2</sup> Auth<sup>1</sup> **Body<sup>1</sup>** Tests Pre Run

```

1  {
2    "mail": "root@test.com",
3    "password": "123456"
4  }

```

Status: 200 OK Size: 191 Bytes Time: 3 ms

**Response** Headers<sup>6</sup> Cookies Results Docs

```

1  {
2    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtYWlsIjoicm9vdEB0ZXN0LmNvbSI6InR5bGU0Ij0iZG1pbiIsIm1hdCI6MTczNjU1MTkwOCwiZXhwIjoxNzY2NTU1NTA4fQ.KGx8WrDvW9r6BAerrEIfOMP20klBvR0buF7V09uH5tY"
3  }

```

- O cliente faz uma requisição, enviando o token no cabeçalho Authorization no formato Bearer <token>.O cliente envia uma requisição contendo o token no cabeçalho Authorization (Bearer <token>);
- O middleware **authenticateToken**:
  - Verifica a validade do token;
  - Em caso de sucesso, permite que o fluxo continue até a função desejada (neste exemplo, a função objetivo);
  - Caso contrário, retorna uma resposta de erro.

Nota: na função **authenticateToken**, o token decodificado é armazenado no objeto **locals** do objeto res, permitindo que as informações sejam acessadas posteriormente sem necessidade de novas decodificações.

The screenshot shows a web browser's developer tools interface. The top bar indicates a GET request to `http://localhost:3101/protected` with a 'Send' button. Below this, the 'Auth' tab is selected, showing a Bearer token. The 'Response' tab is also open, displaying a JSON object with the following structure:

```

1  {
2    "message": "Bem-vindo à rota protegida!",
3    "user": {
4      "decoded": {
5        "mail": "root@test.com",
6        "role": "admin",
7        "iat": 1736551908,
8        "exp": 1736555508
9      }
10   }
11  }

```

The status bar at the top of the response tab shows: Status: 200 OK, Size: 135 Bytes, Time: 5 ms.

```

import express, { NextFunction, Request, Response } from "express";
import jwt from "jsonwebtoken";
import dotenv from "dotenv";

dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;
const SECRET_KEY = process.env.JWT_SECRET || "chave";

app.use(express.json());

app.listen(PORT, () => {
  console.log(`Servidor rodando em http://localhost:${PORT}`);
});

// Middleware para verificar o token
function authenticateToken(req: Request, res: Response, next: NextFunction): void {
  const authHeader = req.headers.authorization;

  if (!authHeader) {
    res.status(401).json({ error: "Token não fornecido." });
  } else {
    const token = authHeader.split(" ")[1];

    try {
      const decoded = jwt.verify(token, SECRET_KEY);

```

```
// Adiciona as informações do token no objeto locals
res.locals = {decoded};
next(); // Permite que a requisição prossiga
} catch (err) {
  res.status(403).json({ error: "Token inválido ou expirado." });
}
}
}

// Rota de login para gerar o token
app.post("/login", (req: Request, res: Response) => {
  const { mail, password } = req.body;

  if (mail === "root@test.com" && password === "123456") {
    const token = jwt.sign({ mail, role: "admin" }, SECRET_KEY, {
      expiresIn: "1h",
    });
    res.json({ token });
  } else {
    res.status(401).json({ error: "Credenciais inválidas." });
  }
});

// Rota protegida
app.get("/protected", authenticateToken, (req: Request, res: Response) => {
  res.json({ message: "Bem-vindo à rota protegida!", user: res.locals });
});
```

O objeto `locals` é uma propriedade do objeto Response que permite o compartilhamento de dados entre os middlewares e as funções subsequentes no processamento da rota.

Benefícios do uso de locals:

- Segurança: os dados armazenados em locals são limitados ao contexto de uma única requisição;
- Eficiência: evita decodificações ou operações redundantes ao permitir o compartilhamento de informações temporárias, como o token decodificado, entre as etapas do processamento da requisição.

Exemplo de uso no código acima: O token decodificado é armazenado em `res.locals` dentro do middleware `authenticateToken`, permitindo que a rota protegida acesse essas informações para personalizar a resposta ao cliente.

## V. Exercícios

Veja os vídeos se tiver dúvidas nos exercícios:

Exercício 1 - <https://youtu.be/kSQhs92vOpk>

Exercício 2 - <https://youtu.be/68nrT3ILpjY>

Exercício 3 - <https://youtu.be/3qSkctRmMaE>

Exercício 4 - <https://youtu.be/cmhsTLMaflA>

**Exercício 1** – Utilize o código a seguir para criar um servidor que roda na porta 3101. Adicionar no código a rota **HTTP GET /um** que envia pelo body um JSON com as propriedades mail e senha. A função objetivo será executada somente se o e-mail e senha forem **abc@teste.com** e **123**, respectivamente. A seguir tem-se exemplos de respostas.

Exemplo de requisição processada pela função **objetivo**:

GET	▼	http://localhost:3101/um
Query	Headers <sup>2</sup>	Auth
Body <sup>1</sup>	Tests	
JSON	XML	Text
Form	Form-encode	
1	{	
2	"mail": "abc@teste.com",	
3	"senha": "123"	
4	}	
Status: 200 OK	Size: 8 Bytes	Time: 3 ms
1	Resposta	

Exemplo de requisição bloqueada pela função **validar**:

GET	▼	http://localhost:3101/um
Query	Headers <sup>2</sup>	Auth
Body <sup>1</sup>	Tests	
JSON	XML	Text
Form		
1	{	
2	"mail": "abc@teste.com",	
3	"senha": ""	
4	}	
Status: 401 Unauthorized	Size: 27 Bytes	
1	{	
2	"error": "Não autorizado"	
3	}	

Observação: o código fornecido não poderá ser alterado.

```
import express, {NextFunction, Request, Response} from "express";

const app = express();
app.use(express.json());

const porta = 3101;
app.listen(
  porta,
  () => console.log(`Rodando na port ${porta}`)
);

const objetivo = (req: Request, res: Response) => {
  res.send("Resposta");
}

const validar = (req: Request, res: Response, next: NextFunction) => {
  const {mail, senha} = req.body;

  if( mail == "abc@teste.com" && senha == "123" ){
    next();
  }
  else{
    res.status(401).send({error: "Não autorizado"});
  }
}
```

```
}  
};
```

**Exercício 2** – Utilize o código a seguir para criar um servidor que roda na porta 3101. Adicionar no código as rotas:

- **HTTP GET /logar** que recebe pelo body um JSON com as propriedades mail e senha. Essas propriedades deverão ser validadas pela função `logar` e retornará o token gerado usando JWT;
- **HTTP GET /dois** que recebe pelo body um JSON com o token recebido na requisição /logar. Essa rota deverá chamar a função `validar`, como middleware, antes de chamar a função `objetivo`.

A seguir tem-se exemplos de resposta.

Exemplo de requisição processada pela função `logar`:

GET `http://localhost:3101/logar` Send

Query Headers <sup>2</sup> Auth **Body <sup>1</sup>** Tests Pre Run

JSON XML Text Form Form-encode GraphQL

```
1 {  
2   "mail": "abc@teste.com",  
3   "senha": "123"  
4 }
```

Status: **200 OK** Size: **157 Bytes** Time: **14 ms** Response ▼

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibWVpbCI6ImFiY0B0ZXN0ZS5jb20iLCJpYXQiOiJlY0B0ZDd9.8NPKSRbFeBeCXUlmZwbFVCMEjagzzsQ835tZSN4JXUI"  
3 }
```

Exemplo de requisição processada pela função `objetivo`:

GET `http://localhost:3101/dois` Send

Query Headers <sup>2</sup> Auth **Body <sup>1</sup>** Tests Pre Run

JSON XML Text Form Form-encode GraphQL

```
1 {  
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwibWVpbCI6ImFiY0B0ZXN0ZS5jb20iLCJpYXQiOiJlY0B0ZDd9.8NPKSRbFeBeCXUlmZwbFVCMEjagzzsQ835tZSN4JXUI"  
3 }
```

Status: **200 OK** Size: **8 Bytes** Time: **14 ms** Response ▼

1 Resposta

Observações:

- O código fornecido não poderá ser alterado;
- Será necessário adicionar as dependências:
  - `npm i jsonwebtoken`
  - `npm i @types/jsonwebtoken -D`

```
import express, { NextFunction, Request, Response } from "express";  
import jwt from "jsonwebtoken";
```

```
const app = express();  
app.use(express.json());
```

```
const porta = 3101;  
app.listen(porta, () => console.log(`Rodando na port ${porta}`));
```

```
const objetivo = (req: Request, res: Response) => {  
  res.send("Resposta");
```

```
};
```

```
const validar = (req: Request, res: Response, next: NextFunction) => {
  const { token } = req.body;
  const secreta = "abc";
  try{
    const decodificado = <any>jwt.verify(token, secreta);
    if (decodificado) {
      res.locals = decodificado;
      next();
    } else {
      res.status(401).send({ error: "Não autorizado" });
    }
  }
  catch(e:any) {
    res.status(401).send({ error: e.message });
  }
};
```

```
const login = (req: Request, res: Response) => {
  const { mail, senha } = req.body;
  if (mail == "abc@teste.com" && senha == "123") {
    const secreta = "abc";
    const token = jwt.sign({ id: 1, mail }, secreta);
    res.json({ token });
  } else {
    res.json({ error: "Dados não conferem" });
  }
};
```

**Exercício 3** – Alterar o código do Exercício 2 para o token ser passado pelo cliente no header da requisição. Utilize Bearer Token.

Exemplo de requisição passando token pelo header:

GET http://localhost:3101/does

Auth: Bearer Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwidmVpY2I6ImFiY0B0ZXN0ZS5jb20iLCJpYXQiOiJlY2OTc4NDMxNDd9.8NPKSRbFeBeCXUlmZwbFVCMEljagzszQ835tZSN4JXUI

Status: 200 OK Size: 8 Bytes Time: 3 ms

Response: 1 Resposta

Exemplo de requisição sem o token:

GET http://localhost:3101/does

Auth: Bearer

enter token

Status: 401 Unauthorized Size: 65 Bytes Time: 3 ms

Response: 1 { 2 "error": "Cannot read properties of undefined (reading 'split') 3 }

**Exercício 4** – Utilize o código a seguir para criar um servidor que roda na porta 3101. O código possui as rotas:

- **HTTP GET /logar** que recebe pelo body um JSON com as propriedades mail e senha. Essas propriedades deverão ser validadas pela função `logar` e retornará o token gerado pelo JWT;
- **HTTP GET /comida** que recebe pelo header um JSON com o token recebido na requisição /logar. Essa rota deverá estar disponível apenas para usuários logados;
- **HTTP GET /veiculo** que recebe pelo header um JSON com o token recebido na requisição /logar. Essa rota deverá estar disponível apenas para usuários com nível “dois”.

Restrição: as funções `validar`, `checarNivel`, `logar`, `carro` e `refeicao` não poderão ser alteradas.

Exemplo de requisição com o token obtido após efetuar o login usando {"mail": "abc@teste.com", "senha": "123"}:

GET

Query Headers **Auth 1** Body 1 Tests

None Basic **Bearer** OAuth 2 NTLM AWS

Bearer Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuaXZlbCI6ImVtliwiaWF0IjoxNjk3ODUwMjAxZQ.chP5iNtP7XmVy-0p0kepXh9zsGC9NWDZ_s2-0PbGUIE
```

Status: **200 OK** Size: 17 Bytes Time: 4 ms

Response Headers 6 Cookies Results Docs

```
1 {
2   "nome": "Alface"
3 }
```

Exemplo de requisição com o token obtido após efetuar o login usando {"mail": "xyz@teste.com", "senha": "abc"}:

Exemplo de requisição com o token obtido após efetuar o login usando {"mail": "abc@teste.com", "senha": "123"}:

GET

Query Headers **Auth 1** Body 1 Tests

None Basic **Bearer** OAuth 2 NTLM AWS

Bearer Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuaXZlbCI6ImVtliwiaWF0IjoxNjk3ODUwMjAxZQ.chP5iNtP7XmVy-0p0kepXh9zsGC9NWDZ_s2-0PbGUIE
```

Status: **401 Unauthorized** Size: 25 Bytes Time: 3 ms

Response Headers 6 Cookies Results Docs

```
1 {
2   "error": "Acesso negado"
3 }
```

Exemplo de requisição com o token obtido após efetuar o login usando {"mail": "xyz@teste.com", "senha": "abc"}:

GET ▼ http://localhost:3101/comida Send

Query Headers <sup>2</sup> **Auth <sup>1</sup>** Body <sup>1</sup> Tests

None Basic Bearer OAuth 2 NTLM AWS

Bearer Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuaXZlbCI6ImRvaXMiLCJpYXQiOiJlYXN0IiwiaWF0Ijoi1NTA1NTk5LnVpF6L8IUJ5tCcnv5GwoM3SBVFGHbk57IlyxWfdq9wiQ
```

Status: 200 OK Size: 17 Bytes Time: 2 ms

Response Headers <sup>6</sup> Cookies Results Docs

```
1 {
2   "nome": "Alface"
3 }
```

GET ▼ http://localhost:3101/veiculo Send

Query Headers <sup>2</sup> **Auth <sup>1</sup>** Body <sup>1</sup> Tests

None Basic Bearer OAuth 2 NTLM AWS

Bearer Token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuaXZlbCI6ImRvaXMiLCJpYXQiOiJlYXN0IiwiaWF0Ijoi1NTA1NTk5LnVpF6L8IUJ5tCcnv5GwoM3SBVFGHbk57IlyxWfdq9wiQ
```

Status: 200 OK Size: 16 Bytes Time: 2 ms

Response Headers <sup>6</sup> Cookies Results Docs

```
1 {
2   "modelo": "Uno"
3 }
```

```
import express, { NextFunction, Request, Response } from "express";
import jwt from "jsonwebtoken";

const app = express();
app.use(express.json());

const porta = 3101;
app.listen(porta, () => console.log(`Rodando na port ${porta}`));

const validar = (req: Request, res: Response, next: NextFunction) => {
  // o token enviado pelo cliente no header da requisição
  const authorization: any = req.headers.authorization;
  const secreta = "abc";
  try {
    // autorização no formato Bearer token
    const [, token] = authorization.split(" ");
    const decodificado = <any>jwt.verify(token, secreta);
    if (decodificado) {
      res.locals = decodificado;
      next();
    } else {
      res.status(401).send({ error: "Não autorizado" });
    }
  } catch (e: any) {
    res.status(401).send({ error: e.message });
  }
};

const checarNivel = (_, req: Request, res: Response, next: NextFunction) => {
  const {nivel} = res.locals;
  if( nivel == "dois" ){
```



```
    next();
  }
  else{
    res.status(401).send({ error: "Acesso negado" });
  }
};

const login = (req: Request, res: Response) => {
  const { mail, senha } = req.body;
  const secreta = "abc";
  if (mail == "abc@teste.com" && senha == "123") {
    const token = jwt.sign({ nivel: "um" }, secreta);
    res.json({ token });
  }
  else if (mail == "xyz@teste.com" && senha == "abc") {
    const token = jwt.sign({ nivel: "dois" }, secreta);
    res.json({ token });
  }
  else {
    res.json({ error: "Dados não conferem" });
  }
};

const carro = (_: Request, res: Response) => {
  res.json({ modelo: "Uno" });
};

const refeicao = (_: Request, res: Response) => {
  res.json({ nome: "Alface" });
};

app.get("/login", login);
app.get("/comida", refeicao);
app.get("/veiculo", carro);
```