



# Aula 04:

Autenticação na API  
com JWT

# JSON Web Token

A autenticação de uma API com JSON Web Token (JWT) é um método usado para verificar a identidade de usuários e proteger o acesso a sistemas.

JWT é um padrão aberto que define uma maneira compacta e segura de transmitir informações entre partes como um objeto JSON.

Para instalá-lo na aplicação rode o comando:

**npm install jsonwebtoken**



# JSON Web Token



**Login:** O cliente (usuário) envia suas credenciais (como nome de usuário e senha) para a API.

**Geração do Token:** Se as credenciais forem válidas, o servidor gera um token JWT assinado com uma chave secreta ou com um par de chaves (pública/privada). Esse token contém informações do usuário, como um identificador (ID), permissões, e uma data de expiração.

**Envio do Token:** O token é enviado de volta ao cliente e armazenado, geralmente em um cookie seguro ou local storage.

**Uso do Token:** Para cada solicitação futura à API, o cliente inclui o token no cabeçalho da requisição (normalmente no campo Authorization com o esquema Bearer).

**Validação do Token:** O servidor valida o token recebido, verificando sua assinatura e outros dados (como validade) antes de permitir o acesso aos recursos da API.



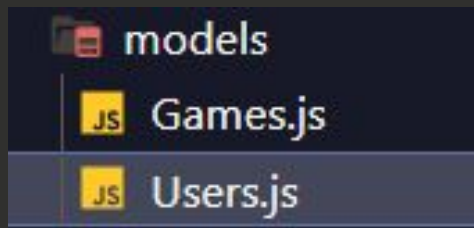


# Aula 04.1:

Cadastro de usuário e login

# Criando o Model de Usuário

Criaremos o modelo que irá representar a entidade **User** em nosso sistema. Para isso, dentro da pasta Models, crie o arquivo **User.js**. Esse arquivo terá a seguinte estrutura:



```
import mongoose from 'mongoose'
```

1 - Importação da biblioteca Mongoose;

```
const userSchema = new mongoose.Schema({  
  name: String,  
  email: String,  
  password: String,  
})
```

2 - Criação de um novo **Schema\***

```
const User = mongoose.model("User",  
userSchema)
```

3 - Criação do model. Aqui informamos que deve ser criado uma coleção que receberá o nome de **users** quando for para o banco de dados.

```
export default User
```

4 - Eportação do módulo.



# Criando o Model de Usuário

Após isso, realize a importação do model no arquivo **index.js**

```
import User from './models/Users.js'
```

Em seguida, rode a aplicação e verifique se a coleção **users** foi criada no banco de dados.



# Cadastrando usuários

Crie o arquivo **userService.js**. Para começarmos a cadastrar usuários na API, criaremos um método chamado **Create()** na classe **userService** que irá inserir novos usuários no banco de dados.

```
1 import User from "../models/Users.js";
2
3 class userService {
4   async Create(name, email, password) {
5     try {
6       const newUser = new User({
7         name,
8         email,
9         password,
10      });
11      await newUser.save();
12    } catch (error) {
13      console.log(error);
14    }
15  }
16 }
17
18 export default new userService();
```

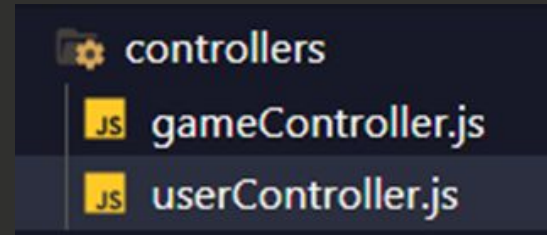


# Cadastrando usuários

Após isso, crie o arquivo **UserController.js**, nele criaremos a constante **createUser** que recebe uma função assíncrona.

Nessa função será coletado os campos vindos do corpo da requisição POST, em seguida será chamado o método **Create()** do service para cadastrar o usuário no banco.

Lembre-se também de **exportar createUser** no final do arquivo.



```
1 import userService from "../services/userService.js";
2 import jwt from "jsonwebtoken";
3 const JWTSecret = "apigamessecret";
4
5 //Cadastrando um Usuário
6 const createUser = async (req, res) => {
7   try {
8     const { name, email, password } = req.body;
9     await userService.Create(name, email, password);
10    res.sendStatus(201); //Código 201 (CREATED)
11   } catch (error) {
12     console.log(error);
13     res.sendStatus(500); // Erro interno do servidor
14   }
15 };
16
17 export default { createUser };
```

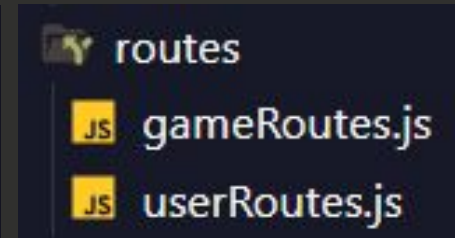




# Cadastrando usuários

Agora devemos apenas criar um novo **endpoint** `"/user"` no arquivo **userRoutes.js**, que receberá a requisição **POST** e chamará o método **createUser()** do **UserController** para tratar a requisição:

```
1 import express from 'express'
2 const userRoutes = express.Router()
3 import UserController from '../controllers/userController.js'
4
5 // Endpoint para cadastrar um Usuário
6 userRoutes.post("/user", UserController.createUser)
7
8 export default userRoutes
```



Feito isso, importe as rotas do **userRoutes** no arquivo principal **index.js**, assim como feito com **gameRoutes**. Agora, testaremos a requisição no **Insomnia** ou **Postman**, enviando uma requisição **POST** para a rota `"/user"`. Será necessário enviar os dados que se deseja cadastrar no corpo da requisição (**BODY**) em formato **JSON**. Envie os campos `"email"` e `"password"` em um JSON. Confira se o usuário é cadastrado.



# Gerando o Token de autenticação

No arquivo **userService.js**. Criaremos mais um método chamado **getOne()** na classe **userService** que irá retornar os dados de um único usuário, quando for realizado o login.

```
1  async getOne(email) {
2    try {
3      const user = await User.findOne({ email: email });
4      return user;
5    } catch (error) {
6      console.log(error);
7    }
8  }
9 }
10
11 export default new userService();
```



Nos próximos slides terá o código do método **loginUser()** para ser inserido no arquivo **UserController.js**. Esse método será responsável por fazer o login do usuário e devolver um token de autenticação JWT.





```
1 // Função para LOGIN do Usuário
2 const loginUser = async (req, res) => {
3   try {
4     const { email, password } = req.body;
5     // E-mail válido
6     if (email !== undefined) {
7       const user = await userService.getOne(email);
8       // Usuário encontrado
9       if (user !== undefined) {
10        // Senha correta
11        if (user.password === password) {
12          jwt.sign(
13            { id: user._id, email: user.email },
14            JWTSecret,
15            { expiresIn: "48h" },
16            (err, token) => {
17              if (err) {
18                res.status(400); // Bad request
19                res.json({ err: "Falha interna" });
20              } else {
21                res.status(200); // OK
22                res.json({ token: token });
23              }
24            }
25          );
26          // Senha incorreta
27        }
28      }
29    }
30  }
31 }
```

⚙ controllers

JS

gameController.js

JS

userController.js

```
20         } else {
21             res.status(200); // OK
22             res.json({ token: token });
23         }
24     }
25 );
26 // Senha incorreta
27 } else {
28     res.status(401); // Unauthorized
29     res.json({ err: "Credenciais inválidas!" });
30 }
31 // Usuário não encontrado
32 } else {
33     res.status(404); // Not Found
34     res.json({ err: "O e-mail enviado não foi enc
35 }
36 // E-mail inválido
37 } else {
38     res.status(400); // Bad request
39     res.json({ err: "O e-mail enviado é inválido."
40 }
41 } catch (error) {
42     console.log(error);
43     res.sendStatus(500); // Internal Server Error
44 }
45 };
46
47 export default { createUser, loginUser };
```



# Gerando o Token de autenticação

Agora devemos apenas criar um novo endpoint `/auth` no arquivo `userRoutes.js`, que receberá a requisição **POST** e chamará o método `loginUser()` do `userController` para tratar a requisição de login:

```
1 // Endpoint para Login do Usuário
2 userRoutes.post("/auth", userController.loginUser)
```

Agora, testaremos a requisição no **Insomnia** ou **Postman** enviando uma requisição **POST** para a rota `/auth`. Envie os campos `email` e `password` em um JSON. A API deve retornar o **token de autenticação**.





# Aula

Middleware de autenticação

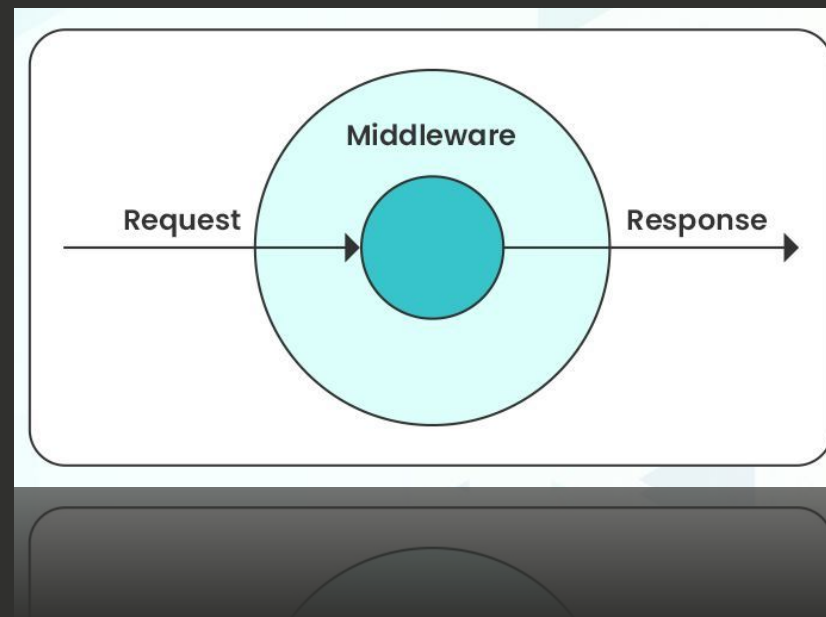
04.2º

# Middleware de autenticação

um middleware é uma função que tem acesso ao objeto de requisição (**req**), ao objeto de resposta (**res**), e à próxima função de middleware no ciclo de solicitação/resposta de uma aplicação.

O middleware pode executar várias tarefas, como:

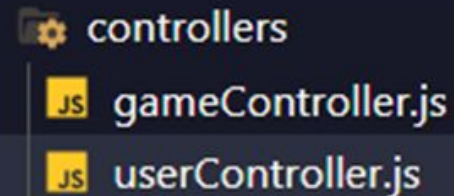
- Executar qualquer código.
- Modificar o objeto de requisição (req) ou resposta (res).
- Encerrar o ciclo de requisição/resposta, enviando uma resposta ao cliente.
- Chamar a próxima função de middleware no stack usando a função **next()**. Se o middleware atual não terminar o ciclo de requisição/resposta, ele deve invocar `next()` para passar o controle para o próximo middleware.



# Middleware de autenticação

O primeiro passo é exportar o **JWTSecret** no arquivo **UserController.js** para podermos usá-lo no middleware de autenticação.

```
1 export default { createUser, loginUser, JWTSecret }
```



controllers


- gameController.js
- UserController.js

Após isso, criaremos nosso middleware de autenticação. O middleware irá verificar se o cliente enviou um **token JWT** (Json Web Token) no cabeçalho da requisição, sob o campo "authorization". Se o token estiver presente e for válido, ele é decodificado, e as informações do usuário (como ID e e-mail) são anexadas ao objeto req. Isso permite que essas informações sejam acessadas nas próximas etapas da requisição. Caso o token seja inválido ou ausente, a requisição é rejeitada com um status **401** (não autorizado), e uma mensagem de erro é enviada ao cliente. Para isso, crie uma pasta com o nome **middleware** na raiz do projeto e em seguida um arquivo dentro da pasta chamado **Auth.js**. O código deste arquivo está nos próximos slides.





```
1 import jwt from "jsonwebtoken";
2 import userController from "../controllers/userController.js";
3
4 // Função de Autenticação com JWT - Json Web Token
5 const Authorization = (req, res, next) => {
6   const authToken = req.headers["authorization"];
7   if (authToken !== undefined) {
8     const bearer = authToken.split(" ");
9     var token = bearer[1];
10    jwt.verify(token, userController.JWTSecret, (err, data) => {
11      if (err) {
12        res.status(401);
13        res.json({ err: "Token inválido!" });
14      } else {
15        req.token = token;
16        req.loggedUser = {
17          id: data.id,
18          email: data.email,
19        };
20        next();
21      }
22    });
23  } else {
24    res.status(401);
25    res.json({ err: "Token inválido" });
26  }
27 };
28
29 export default { Authorization };
```

 middleware

 Auth.js

# Middleware de autenticação

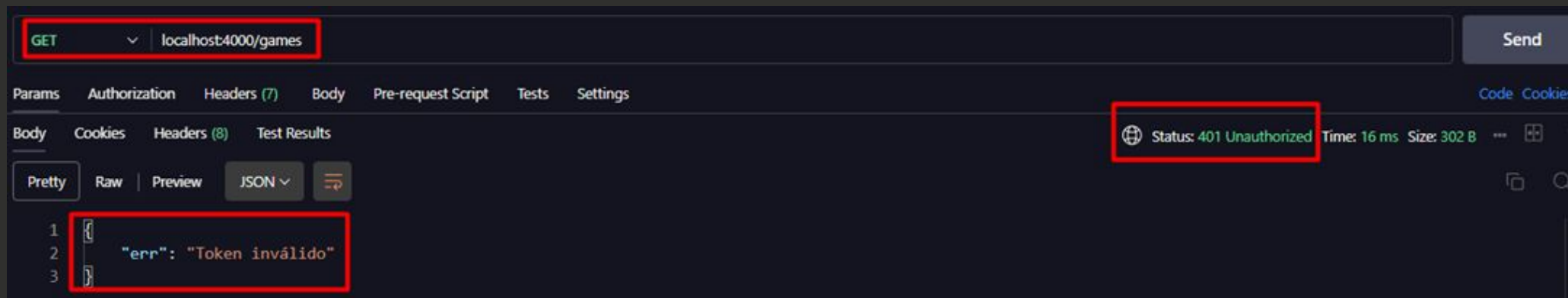
Após isso, devemos importar o middleware de autenticação no arquivo **gameRoutes.js**, e adicioná-lo em todas as rotas, conforme mostra a imagem abaixo:

```
1 import express from 'express'
2 const gameRoutes = express.Router()
3 import gameController from '../controllers/gameController.js'
4 import Auth from '../middleware/Auth.js'
5
6 // Endpoint para listar todos os Games
7 gameRoutes.get("/games", Auth.Authorization, gameController.getAllGames)
8
9 // Endpoint para cadastrar um Game
10 gameRoutes.post("/game", Auth.Authorization, gameController.createGame)
11
12 // Endpoint para deletar um Game
13 gameRoutes.delete("/game/:id", Auth.Authorization, gameController.deleteGame)
14
15 // Endpoint para alterar um Game
16 gameRoutes.put("/game/:id", Auth.Authorization, gameController.updateGame)
17
18 // Endpoint para listar um único Game
19 gameRoutes.get("/game/:id", Auth.Authorization, gameController.getOneGame)
20
21 export default gameRoutes
```



# Middleware de autenticação

Por fim, realizaremos os testes no **Insomnia** ou **Postman**. No exemplo abaixo será usado o Postman. Agora, ao tentarmos acessar a rota **“/games”** é necessário informar o token de autenticação, caso contrário a API retornará o código de status **401 (Não autorizado)**.



Agora para ter acesso a listagem de games precisamos primeiramente realizar o login enviando **email** e **password** para a rota **“/auth”**.



# Middleware de autenticação

Ao realizarmos o login, a API nos retorna o token de autenticação:



The screenshot displays a REST client interface with the following details:

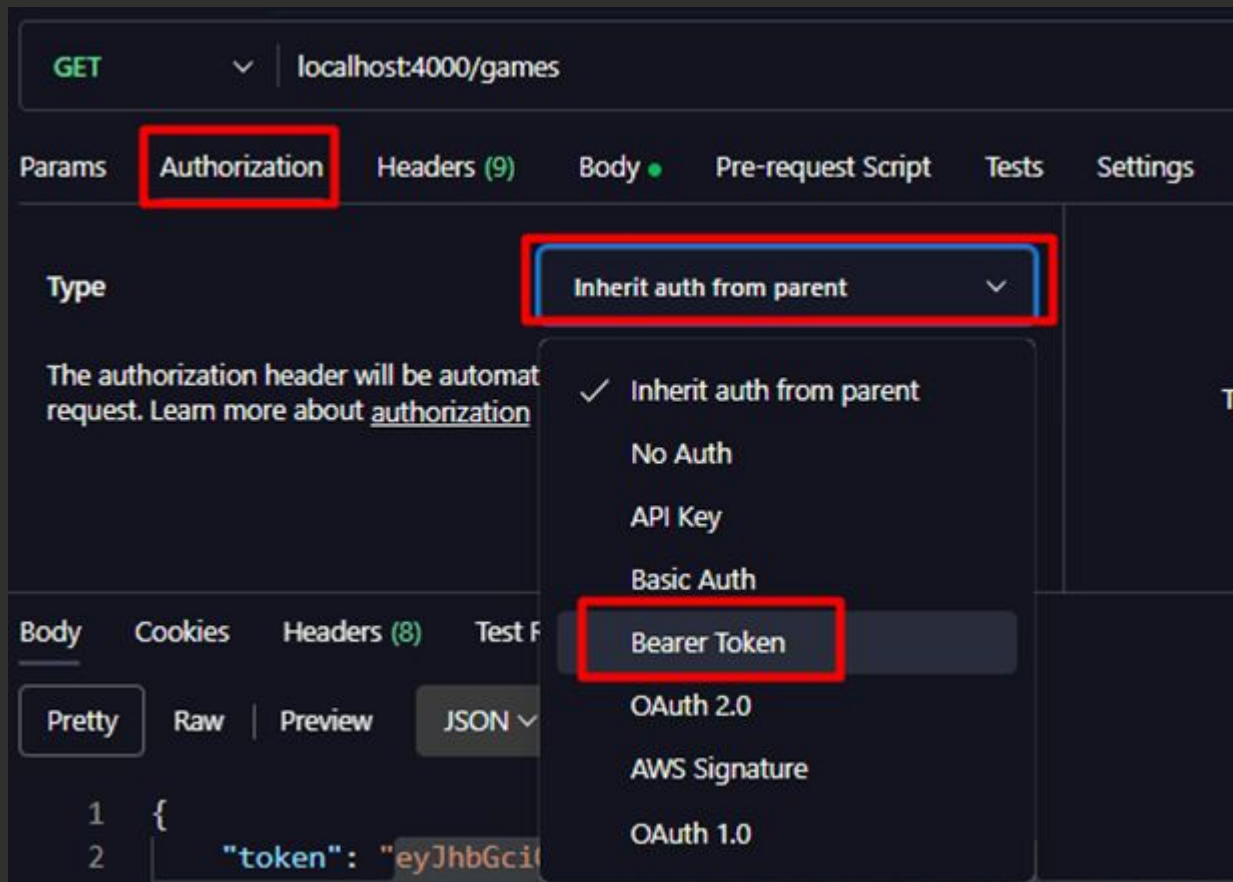
- Method:** POST
- URL:** localhost:4000/auth
- Body Type:** raw (selected)
- Request Body (JSON):**

```
{  "email": "diego@email.com",  "password": "123456"}
```
- Status:** 200 OK
- Response Body (JSON):**

```
{  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6IjY2NDUwYj1hYmJkMTIyOTU3ZDMwZGMxOSIsImVtYWlsIjo1ZG1lZ29AZW1haWwY29tIiwiaWF0IjoxNzE1NTQ2Mzk5LCJleHAiOjE3MTU3MTkxOT19LyaepUS00gPyFLCSv8BQoLc0-5oy7b4ZhdpwIFY2qS3o"}
```

# Middleware de autenticação

Agora devemos inserir o token no cabeçalho da requisição, informando o tipo do token que é o **Bearer Token**. Segue logo abaixo um exemplo de como fazer isso no Postman.

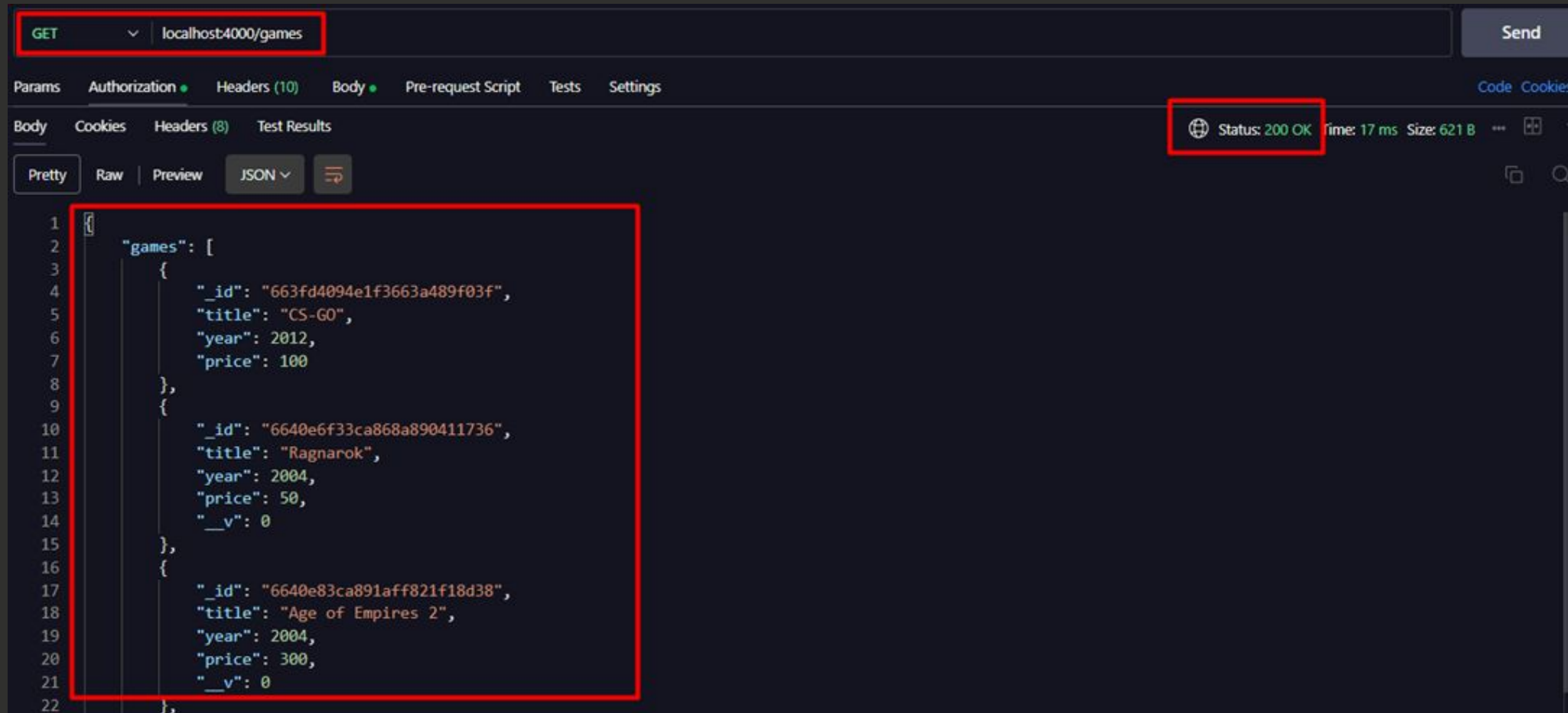






# Middleware de autenticação

Com a devida autenticação, agora a API nos permite acessar o recurso normalmente e temos acesso a lista de games. Isso serve também para as outras requisições da API.



```
GET localhost:4000/games

Status: 200 OK Time: 17 ms Size: 621 B

{
  "games": [
    {
      "_id": "663fd4094e1f3663a489f03f",
      "title": "CS-GO",
      "year": 2012,
      "price": 100
    },
    {
      "_id": "6640e6f33ca868a890411736",
      "title": "Ragnarok",
      "year": 2004,
      "price": 50,
      "__v": 0
    },
    {
      "_id": "6640e83ca891aff821f18d38",
      "title": "Age of Empires 2",
      "year": 2004,
      "price": 300,
      "__v": 0
    }
  ]
}
```





# Aula 04:

Autenticação na API  
com JWT