Desenvolvimento Backend

Aula 13 - Autenticando usuários na API

Material Didático do Instituto Metrópole Digital - IMD

Apresentação

Nesta aula, veremos como fazer a autenticação do usuário. Veremos ainda algumas boas práticas e algumas partes que devem ser melhoradas na aplicação.

Objetivos

- Conhecer como criptografar dados
- Aprender como fazer a autenticação de usuários
- Entender como manter um usuário autenticado
- Aprender a gerar um *token*

Criptografando a senha no banco de dados

Link do video da aula: https://youtu.be/6OOghHkgztE

Atualmente, visitando a tabela do banco de dados dos usuários, é visível que a senha é armazenada diretamente, ou seja, o conteúdo do campo "senha" no banco é exatamente igual ao que foi digitado pelo usuário do sistema. Isso não é usual, visto que é uma informação confidencial, devendo ser de conhecimento apenas do dono da senha.

Criptografando a senha

Para resolver esse problema, é importante que antes de ser armazenada no banco, a senha seja encriptada. Então, recorremos à biblioteca *bcrypt*.

Para instalar a biblioteca, utilize o comando:

```
npm install bcrypt
```

Agora, antes de salvar o usuário no banco, deve-se criptografar a senha digitada por ele.

Navegando até a rota *post*, no arquivo de '*usuario.rota.js*', percebemos que as informações estão sendo armazenadas exatamente como veio na requisição. Para criptografar a senha da maneira correta, acompanhe a aula e observe como são feitas as alterações.

Ao final, sua rota deve ficar como a rota abaixo:

```
router.post("/", async (req, res) => {
  const senha = req.body.senha;
  const salt = await bcrypt.genSalt(10);
  const senhaCriptografada = await bcrypt.hash(senha, salt);
  const usuario = { email: req.body.email, senha: senhaCriptografada
};
  const usuarioObj = await Usuario.create(usuario);
  res.json({ msg: "Usuário adicionado com sucesso!", userId:
usuarioObj.id });
});
```

Autenticação

Link do video da aula: https://youtu.be/KJS_U8MtbN8

Atualmente, já temos a senha salva no banco de dados. Agora, vamos ter de verificar se o e-mail e a senha que o usuário digitou estão corretos.

Autenticação do usuário

Para fazer a autenticação, precisamos criar uma rota de *login* para o usuário. Então, no arquivo '*usuario.rota.js*', será criada uma rota do tipo *post* como se vê abaixo:

```
router.post("/login", async (req, res) => {
  const email = req.body.email;
  const senha = req.body.senha;

  const usuario = await Usuario.findOne({
    where: {
      email: email
    }
  });

  if (usuario && (await bcrypt.compare(senha, usuario.senha))) {
        console.log('Sucesso')
  } else {
    console.log('Falha ao autenticar. Usuário ou senha inválidos')
  }
});
```

Para testar se é possível encontrar o e-mail e a senha no banco, é preciso criar uma nova rota de usuário no arquivo oculto '.usuario.http'. Então, abra o arquivo e

implemente a rota abaixo:

```
POST http://localhost:8080/api/usuarios/login
Content-Type: application/json

{
    "email": "turing@gmail.com",
    "senha": "senha"
}
```

Os campos de '*email*' e 'senha' passados no *body* devem ser os que o usuário deseja fazer a autenticação.

Entendendo o Json Web Token (JWT)

Link do video da aula: https://youtu.be/4MnAtAXIQzk

Já está sendo feita a autenticação do usuário, entretanto, é preciso entender como manter o usuário autenticado.

Conhecendo o JWT

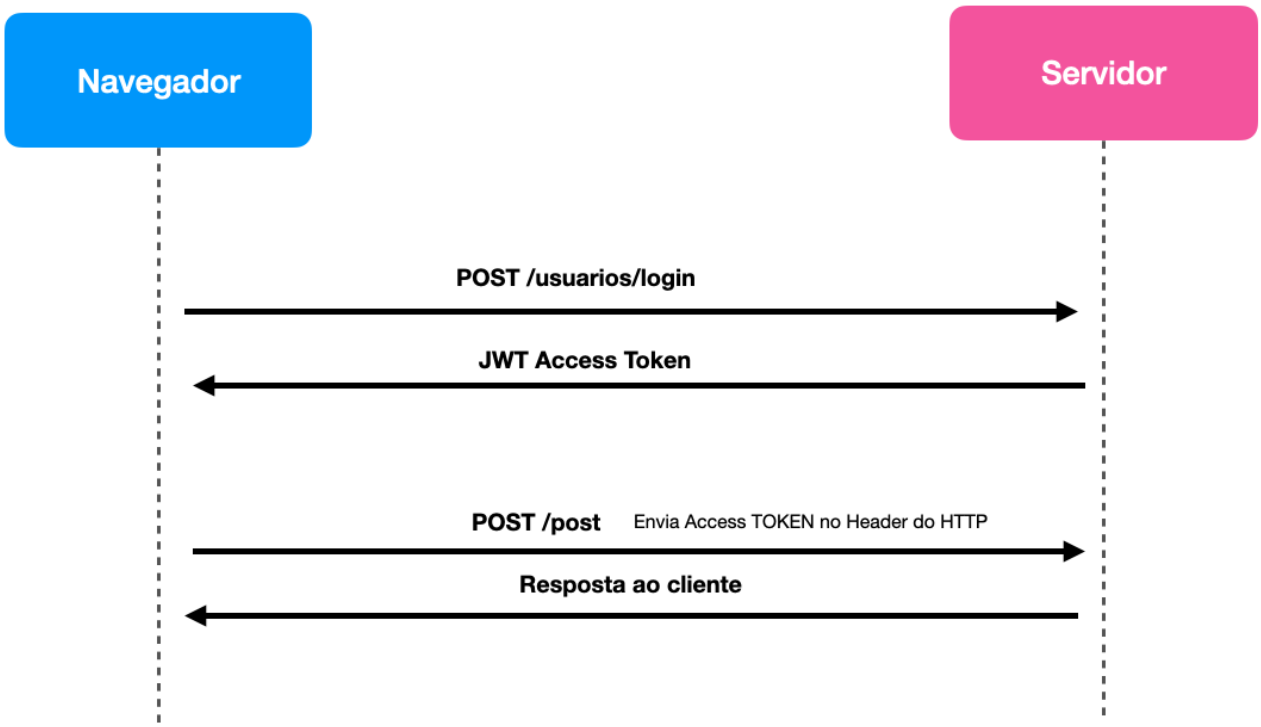
Json Web Token (JWT) é uma tecnologia utilizada para realizar autenticação entre duas partes através de um token. A ideia é que, ao autenticar, o usuário retorne um *token*, que é uma sequência de caracteres que guarda um conjunto de informações que estão codificadas, dentre elas a informação de que o usuário está autenticado.

Figura 1 - Exemplo de Token JWT

Como funciona a aplicação usando JWT

A imagem abaixo apresenta o funcionamento da tecnologia. Acompanhe a aula para mais detalhes.

Figura 2 - Funcionamento JWT



Acessando o site do JWT é possível visualizar uma característica muito interessante: o token *JWT* tem três partes, sendo elas o cabeçalho, a informação e uma assinatura gerada no servidor. Podemos ver essa diferenciação por cores abaixo:

Figura 3 - Divisão do token

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey
JzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpva
G4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKx
wRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Essa divisão é beneficial na autenticação, isso porque no processo de gerar a assinatura no servidor é preciso de uma *secret* que mais ninguém conhece. Portanto, qualquer adulteração feita no *token* vai gerar uma invalidação, pois irá gerar uma assinatura diferente.

Gerando Token JWT

Link do video da aula: https://youtu.be/3cnMifiwF7I

Agora, já conhecendo um pouco sobre o JWT, iniciaremos a implementação incrementando a rota de *login*.

Gerando um *token* JWT

No arquivo '*usuario.rota.js*' existe a rota de login, que está retornando uma mensagem de sucesso ou falha. No entanto, a ideia é que ao dar tudo certo, seja retornado um *token*.

Para isso acontecer, antes de mais nada é preciso instalar duas bibliotecas. No terminal, digite:

```
npm i jsonwebtoken
```

e

```
npm i dotenv
```

Após isso, importe a biblioteca *JWT* através da linha:

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

Logo em seguida, modifique a rota de *login* para em caso de sucesso retornar um *token*. Sua rota deverá ficar como abaixo:

```
router.post("/login", async (req, res) => {

  const email = req.body.email;
  const senha = req.body.senha;

  const usuario = await Usuario.findOne({
    where: {
      email: email,
    },
  });

  if (usuario && (await bcrypt.compare(senha, usuario.senha))) {
    const payload = {
      sub: usuario.id,
      iss: "imd-backend",
      aud: "imd-frontend",
      email: usuario.email,
    };
    const token = jwt.sign(payload, process.env.ACCESS_TOKEN_SECRET)
    res.json({ accessToken: token })
  } else {
```

```
    res.status(403).json({ msg: "usuário ou senha inválidos" })
  }
});
```

Essa variável de ambiente '*ACCESS_TOKEN_SECRET*' precisa ser definida. Para isso, com o auxílio da biblioteca *dotenv*, crie um arquivo oculto '*.env*' e atribua um valor aleatório a ela.

Exemplo de definição:

```
ACCESS_TOKEN_SECRET=e74e41e5561cd01e0d2b4e0a3183f8473e5288dc43c44fe3
2879f61a890b2491a6484be7ea2615a49ad52476d6dd3894a0b1201e7224de07b44f
b1dc590ab181
```

Por fim, é preciso carregar o *dotenv* na inicialização. Então, no arquivo '*index.js*' adicione a seguinte linha nas importações:

```
    require('dotenv').config()
```

Validando Token JWT

Link do video da aula: https://youtu.be/kOaGoFCAEHs

Agora que já estamos gerando o *token* JWT, é preciso validar. A ideia é proteger algumas rotas para que só usuários autenticados possam utilizá-las.

Criando um *middleware* de validação

Na pasta de *middleware*, crie um arquivo, nomeando-o de '*autenticacao.mid.js*' e, acompanhando a aula, implemente o código abaixo:

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

function autenticar(req, res, next){

    const auth = req.headers["authorization"]
    const token = auth && auth.split(' ')[1]
    if (!token){
        return res.sendStatus(401)
    }else{
        jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err,
paylod) => {
```

```
            if (err) return res.sendStatus(403)
            req.user = paylod
            next()
        })
    }
}

module.exports = autenticar
```

Utilizando o *middleware*

Após criado, para utilizar o *middleware*, faça a importação no arquivo '*post.rota.js*' através da linha:

```
const autenticar = require('../middleware/autenticacao.mid')
```

E após isso, utilize a função de autenticar nas rotas que fazem alteração nos *posts*, como abaixo:

```
router.post('/', autenticar, upload.single('foto'))
router.post('/', autenticar, postMid)
router.put('/', autenticar, postMid)
```

Criando um *post*

Agora, quando o usuário desejar fazer um novo *post*, deverá estar autenticado. Para testar isso, altere a requisição de adição de um *post* no arquivo ''*.post.http* inserindo novas informações.

A requisição deverá ficar como se vê a seguir:

```
POST http://localhost:8080/api/posts
Content-Type: application/json
Authorization: Bearer 'Token'

{
    "titulo": "Programando em JS Backend",
    "texto": "Título do texto...",
    "userId": 3
}
```

A informação ''*Token* deverá ser substituída pelo *token*, que será retornado ao usuário fazer a autenticação.

Limitando o tempo de autenticação

Até o momento, ao conseguir esse *token*, ele fica para sempre válido e isso não é uma boa prática. Então, ao gerar o *token* na rota de *login*, é interessante passar um tempo de vida para essa informação.

Então, no momento em que estiver gerando o *token*, passe o parâmetro *expiresIn* com o tempo de vida dele. A alteração é feita dessa forma:

```
const token = jwt.sign(payload, process.env.ACCESS_TOKEN_SECRET,
{expiresIn: '40s'})
```

Resumo

Nesta aula, foram vistos conceitos e tecnologias de autenticação de usuários. Dentre esses conceitos, foi visto como fazer criptografia, o que era e como gerar um *token*, como realizar a autenticação e como manter o usuário autenticado.