

SOFTWARE SEM SEGURANÇA NÃO SERVE!

Desenvolvedor: André Luis Gonçalves Carvalho

Matrícula: 202203185403

Contextualização

1. Abra o código-fonte fornecido acima na IDE ou editor;
2. Refatore o método de criptografia utilizado atualmente, substituindo a geração do "session-id" por um outro mecanismo de segurança, como tokens JWT;
3. Refatore a arquitetura da API, para que o token (atualmente representado pelo "session-id") não seja trafegado via URI, mas através do header da requisição;
4. A cada requisição recebida pela API, valide o token de segurança, incluindo a identidade do usuário, data/hora de expiração do mesmo, etc.;
5. Inclua, em todos os endpoints, controle de acesso a recursos baseado no perfil do usuário. Garante que, à exceção do endpoint de login, todos os demais sejam acessados apenas por usuários com perfil 'admin';
6. Para testar a implementação do item anterior, crie um novo endpoint que permita a recuperação dos dados do usuário logado. Tal método não deverá conter o controle de acesso limitado ao perfil 'admin';
7. Refatore o método que realiza a busca de contratos no banco, tratando os parâmetros recebidos contra vulnerabilidades do tipo "Injection". Para isso você poderá utilizar bibliotecas de terceiros, expressões regulares ou outro mecanismo que garanta o sucesso do processo em questão;
8. Salve o código e coloque a API para ser executada;
9. Utilizando um cliente (Insomnia, Postman ou outro de sua preferência), realize testes na API, garantindo que todos os pontos acima foram tratados

Estrutura

\ - Diretório raiz, com o index.js modificando o código original fornecido e o package.json.

Controller - Pasta de Controladores, contém o UserController implementados os métodos, buscar por nome de usuário e buscar todos.

Models - Possui o modelo do objeto User


Repositories - Possui os dados dos usuários em forma de Array de Usuários

UseCases\JwtToken - Possui as implementações de gerador, validador e decodificador de token jwt.




UserCases\Login - Possui a implementação do método de login, que recebe usuário e senha, utiliza o controller de usuários, para validar o login e retorna o token jwt gerado pelo gerador de token.

Procedimentos

1. Cópia do código original.
2. Criação de um **package.json**
3. Criação de um **.gitignore**
4. Execução do **npm install** para instalação das dependências.
5. Criação de um gerador de token, com base nos parâmetros do projeto.

```
UseCases > JwtToken >  generatetoken.js > [⌘] <unknown>
1  const jwt = require('jsonwebtoken');
2  const SECRET_KEY = process.env.SECRET_KEY || 'test_key';
3
4  function GenerateToken(user) {
5      const payload = {
6          id: user.username,
7          role: user.perfil
8      }
9  }
10
11  const token = jwt.sign(payload, SECRET_KEY, {expiresIn: '1m'});
12  return token
13
14  module.exports = GenerateToken
```

6. Criação de um middleware validador de token.

```
UseCases > JwtToken >  jwtauth.js >  jwtAuth >  run
1  const jwt = require('jsonwebtoken');
2  const SECRET_KEY = process.env.SECRET_KEY || 'test_key';
3
4  class jwtAuth {
5      constructor(role) {
6          this.role = role;
7      }
8
9      run(req, res, next) {
10         let authorization = req.headers.authorization;
11         if (!authorization) {
12             return res.status(401).json({
13                 message: 'No Authorization Header'
14             });
15         }
16         try {
17             let token = authorization.split('Bearer ')[1];
18             if (!token) {
19                 return res.status(401).json({
20                     message: 'Invalid Token Format'
21                 });
22             }
23             let decode = jwt.verify(token, SECRET_KEY);
24             if (decode.role == this.role) {
25                 return next();
26             }
27         } catch (error) {
28             if (error instanceof jwt.TokenExpiredError) {
```

7. Extração dos Dados de usuário para a pasta repositories, como InMemoryData para testes.

```
Repositories > InMemoryUsersData.js > ...
1  const User = require("../Models/User")
2  let Users = [
3    { "username" : "user", "password" : "123456", "id" : 123, "email" : "user@dominio.com", "perfil": "user"},
4    { "username" : "admin", "password" : "123456789", "id" : 124, "email" : "admin@dominio.com", "perfil": "admin"},
5    { "username" : "colab", "password" : "123", "id" : 125, "email" : "colab@dominio.com", "perfil": "user"},
6  ]
7
8  let InMemoryUsersData = new Array(User)
9  for (u in Users) {
10    InMemoryUsersData.push(new User(Users[u].username, Users[u].password, Users[u].id, u.email, Users[u].perfil))
11  }
12
13  module.exports = InMemoryUsersData
```

8. Criação de “Model” para User

```
Model > User.js > User
1  class User {
2    constructor(username, password, id, email, perfil) {
3      this.username = username;
4      this.password = password;
5      this.id = id;
6      this.email = email;
7      this.perfil = perfil;
8    }
9  }
```

9. Criação de um Controller para obter Users

```
Controller > UserController.js > UserController > constructor
1  class UserController{
2    constructor() {
3      // Carregamento da InMemoryUsersData para teste do Trabalho.
4      const Users = require("../Repositories/InMemoryUsersData")
5    }
6
7    findUserByUsername(username){
8      let userData
9      for (User in Users) {
10        userData = User.username == username ? User : ""
11      }
12      return userData
13    }
14
15    // Outros métodos poderiam ter sido desenvolvidos aqui, como addUser, deleteUser, updateUser, porém o exercício não fez a implementação necessária.
16
17  }
```

10. Criação de uma função de login, retornando o token em caso de sucesso.

```
UseCases > Login > login.js > login
1  const UserController = require('../../Controller/UserController')
2  const User = require('../../Models/User');
3  const generateToken = require('../JwtToken/generatetoken');
4
5  function login(username, password){
6      let userController = new UserController();
7      let user = userController.findUserByUsername(username)
8      if(user) {
9          if(user.password == password) {
10             let token = generateToken(user)
11             return token
12          } else {
13             console.log("Usuario e/ou Senha Incorretos")
14          }
15      } else {
16          console.log("Usuario não existe")
17      }
18  }
19
20  module.exports = login
```

11. Implementação do novo login na API

```
app.post('/api/auth/login', (req, res, next) => {
  const credentials = req.body
  let token = login(credentials?.usuario, credentials?.password)
  if(token) {
    res.json("Authorization: Bearer "+token)
  } else {res.status(401).json("Falha ao Efetuar Login")}
})
```

12. Restringindo rotas com Middleware

```
const checkAdmin = new jwtAuth("admin")
app.use(checkAdmin.run.bind(checkAdmin))
```

13. Criando Função para buscar dados do usuário logado.

```
app.get('/api/auth/unrestricted', (req, res, next) => {
  let decoded = jwtDecode(req, res)
  let userController = new UserController()
  let data = userController.findUserByUsername(decoded.id)
  res.status(200).json("Access Granted, user Data:"+JSON.stringify(data))
}) // Passo 6, Novo endpoint para recuperação de dados do usuário não rest
```

14. Adição de auth ao endereço e remoção do sessionid na rota /api/users

```
app.get('/api/auth/users', (req, res) => {
  let userController = new UserController()
  data = userController.getAll()
  res.status(200).json("Usuários: "+JSON.stringify(data))
})
```

15. Refatoração contra SQL Injection usando Express-validator.

```
app.get('/api/auth/contracts/:empresa/:inicio', [
  param('empresa')
    .trim()
    .isLength({ min: 1 })
    .withMessage('Empresa não pode estar vazio')
    .escape(),
  param('inicio')
    .isDate({ format: 'DD-MM-YYYY' })
    .withMessage('Formato Invalido. Use DD-MM-YYYY.'], (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }
    const empresa = req.params.empresa;
    const dtInicio = req.params.inicio;
    const result = getContracts(empresa, dtInicio);
    if(result)
      res.status(200).json({ data: result })
    else
      res.status(404).json({data: 'Dados Não encontrados'})
  })
```

Teste das rotas

http://localhost:3000/api/auth/login - Efetua Login e Devolve Token

ETag: W/"bd-vt8ahh0dahzz1xRpbf8Eu+zxtPs"
Date: Tue, 29 Oct 2024 06:44:01 GMT
Connection: keep-alive
Keep-Alive: timeout=5

"Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6ImFkbWluliwicm9sZSI6ImFkbWluliwiaWF0IjoxI

http://localhost:3000/api/auth/restricted - Testa a restrição de rotas

200 OK

HISTORY

ASSERTIONS

HTTP

DESCRIPTION

HTTP/1.1 200 OK
X-Powered-By: Express
Date: Tue, 29 Oct 2024 06:48:33 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 16
ETag: W/"10-HSY9P2D2HFKh6GyPqQK7KJrvYF4"

"Access Granted"

http://localhost:3000/api/auth/unrestricted - Recupera os dados do usuário

200 OK

HISTORY

ASSERTIONS

HTTP

DESCRIPTION

HTTP/1.1 200 OK
X-Powered-By: Express
Date: Tue, 29 Oct 2024 06:50:35 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 111
ETag: W/"6f-6MYV38MVH97AeKYx00mRFvj2LLs"

"Access Granted, user Data:{\"username\":\"admin\",\"password\":\"123456789\",\"id\":124,\"perfil\":\"admin\"}"

http://localhost:3000/api/auth/users - Obtem todos os usuários do sistema

200 OK

HISTORY

ASSERTIONS

HTTP

Content-Type: application/json; charset=utf-8

Content-Length: 48

ETag: W/"30-BYfRTxsEs2LViJ5EEExC80bR/ugA"

Date: Tue, 29 Oct 2024 06:46:16 GMT

Connection: keep-alive

Keep-Alive: timeout=5

"Usuários: [null,\"user\",\"admin\",\"colab\"]"

http://localhost:3000/api/auth/contracts/Microsoft/20-04-2020

Response

200 OK

HISTORY

ASSERTIONS

HTTP

DESCRIPTION

Content-Type: application/json; charset=utf-8

Content-Length: 11

ETag: W/"b-EFAIOux7Kcr/ZEgGkn2r+oFAbu4"

Date: Tue, 29 Oct 2024 06:54:34 GMT

Connection: keep-alive

Keep-Alive: timeout=5

{"data":[]}

Rotas restritas

Sem Token

response

401 Unauthorized

HISTORYASSERTIONSHTTP

Content-Type: application/json; charset=utf-8

Content-Length: 37

ETag: W/"25-9mD07RywwZuNuWARYFpjKScwQfg"

Date: Tue, 29 Oct 2024 06:55:45 GMT

Connection: keep-alive

Keep-Alive: timeout=5

{"message": "No Authorization Header"}

Token Invalido

401 Unauthorized

HISTORYASSERTIONSHTTP

Content-Type: application/json; charset=utf-8

Content-Length: 51

ETag: W/"33-CenMtZzj51AFIUydqzqoTjTbGJU"

Date: Tue, 29 Oct 2024 06:56:56 GMT

Connection: keep-alive

Keep-Alive: timeout=5

{"message": "Invalid Token", "error": "invalid token"}

Token Expirado

401 Unauthorized

HISTORYASSERTIONSHTTP

Content-Type: application/json; charset=utf-8

Content-Length: 51

ETag: W/"33-8nhNHT31QYvOVetZPdwi0ImKmNE"

Date: Tue, 29 Oct 2024 06:59:23 GMT

Connection: keep-alive

Keep-Alive: timeout=5

{"message": "Session Expired", "error": "jwt expired"}