



# Estácio

**Campus:** Rua Manoel João Gonçalves, 410/412 – Alcântara CEP: 24711-080

**Curso:** Desenvolvimento Full-Stack

**Disciplina:** RPG0035 - SOFTWARE SEM SEGURANÇA NÃO SERVE!

**Turma:** 9001

**Semestre letivo:** 2024.4 FLEX

**Integrante:**

Nome: Samir Campos Lima

Matrícula: 2022.11.47141-1

**Link do repositório no GIT:** [samircamposlima/RPG0035---SOFTWARE-SEM-SEGURAN-A-N-O-SERVE-](https://github.com/samircamposlima/RPG0035---SOFTWARE-SEM-SEGURAN-A-N-O-SERVE-)

# Missão Prática | Software sem segurança não serve

## Código-fonte a corrigir:

```
const express = require('express')
const bodyParser = require('body-parser')
const crypto = require('crypto')

const app = express()

app.use(bodyParser.json())

const port = process.env.PORT || 3000
app.listen(port, () => {
  console.log(`Server is running on port ${port}`)
})

//Endpoint para login do usuário
// Dados do body da requisição: {"username" : "user", "password" : "123456"}
// Verifique mais abaixo, no array users, os dados dos usuários existentes na app
app.post('/api/auth/login', (req, res) => {
  const credentials = req.body

  let userData;
  userData = doLogin(credentials)

  if(userData){
    //cria o token que será usado como session id, a partir do id do usuário
    const dataToEncrypt = `{"usuario_id":${userData.id}}`;
    const bufferToEncrypt = Buffer.from(dataToEncrypt, "utf8");
    hashString = encrypt(bufferToEncrypt)
  }

  res.json({ sessionid: hashString })
})

//Endpoint para demonstração do processo de quebra da criptografia da session-
id
gerada no login
// Esse endpoint, e consequente processo, não deve estar presente em uma API
oficial,
```

```
//      aparecendo aqui apenas para finalidade de estudos.
app.post('/api/auth/decrypt/:sessionid', (req, res) => {
  const sessionid = req.params.sessionid;
  //const decryptedSessionid = decryptData(sessionid);
  const decryptedSessionid = decrypt(sessionid);

  res.json({ decryptedSessionid: decryptedSessionid })
})

//Endpoint para recuperação dos dados de todos os usuários cadastrados
app.get('/api/users/:sessionid', (req, res) => {
  const sessionid = req.params.sessionid;
  const perfil = getPerfil(sessionid);

  if (perfil !== 'admin' ) {
    res.status(403).json({ message: 'Forbidden' });
  }else{
    res.status(200).json({ data: users })
  }
})

//Endpoint para recuperação dos contratos existentes
app.get('/api/contracts/:empresa/:inicio/:sessionid', (req, res) => {
  const empresa = req.params.empresa;
  const dtInicio = req.params.inicio;
  const sessionid = req.params.sessionid;

  const result = getContracts(empresa, dtInicio);
  if(result)
    res.status(200).json({ data: result })
  else
    res.status(404).json({data: 'Dados Não encontrados'})
})

//Outros endpoints da API
// ...

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//
/////

//Mock de dados
const users = [
  {"username" : "user", "password" : "123456", "id" : 123, "email" :
"user@dominio.com", "perfil": "user"},
  {"username" : "admin", "password" : "123456789", "id" : 124, "email" :
"admin@dominio.com", "perfil": "admin"},
  {"username" : "colab", "password" : "123", "id" : 125, "email" :
"colab@dominio.com",
"perfil": "user"},
]

//APP SERVICES
function doLogin(credentials){
  let userData
  userData = users.find(item => {
```

```

    if(credentials?.username === item.username && credentials?.password ===
item.password)
        return item;
    });
    return userData;
}

// Gerando as chaves necessárias para criptografia do id do usuário
// Nesse caso, a palavra-chave usada para encriptação é o nome da empresa
detentora do software em questão.
const secretKey = 'nomedaempresa';
function encrypt(text) {
    const cipher = crypto.createCipher('aes-256-cbc', secretKey);
    let encrypted = cipher.update(text, 'utf8', 'hex');
    encrypted += cipher.final('hex');
    return encrypted;
}

// Função de exemplo para demonstrar como é possível realizar a quebra da chave
gerada (e usada como session id),
// tendo acesso ao algoritmo e à palavra-chave usadas na encriptação.
function decrypt(encryptedText) {
    const decipher = crypto.createDecipher('aes-256-cbc', secretKey);
    let decrypted = decipher.update(encryptedText, 'hex', 'utf8');
    decrypted += decipher.final('utf8');
    return decrypted;
}

//Recupera o perfil do usuário através da session-id
function getPerfil(sessionId){
    const user = JSON.parse(decrypt(sessionId));
    //varre o array de usuarios para encontrar o usuário correspondente ao id
    obtido da
    sessionId
    const userData = users.find(item => {
        if(parseInt(user.usuario_id) === parseInt(item.id))
            return item;
    });
    return userData.perfil;
}

//Classe fake emulando um script externo, responsável pela execução de queries
no
banco de dados
class Repository{
    execute(query){
        return [];
    }
}

//Recupera, no banco de dados, os dados dos contratos
// Metodo não funcional, servindo apenas para fins de estudo
function getContracts(empresa, inicio){
    const repository = new Repository();
    const query = `Select * from contracts Where empresa = '${empresa}' And
data_inicio = '${inicio}'`;

```

```
const result = repository.execute(query);  
  
return result;  
}
```

## - Procedimentos

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;

### Código-fonte corrigido:

```
const express = require('express');
const bodyParser = require('body-parser');
const jwt = require('jsonwebtoken');
const app = express();

app.use(bodyParser.json());

const port = process.env.PORT || 3000;
const SECRET_KEY = 'my_secret_key'; // Substitua por uma chave mais segura em produção

const users = [
  { username: 'user', password: '123456', id: 123, email: 'user@dominio.com', perfil: 'user' },
  { username: 'admin', password: '123456789', id: 124, email: 'admin@dominio.com', perfil: 'admin' },
  { username: 'colab', password: '123', id: 125, email: 'colab@dominio.com', perfil: 'user' },
];

// Middleware para validação de token
defaultauthMiddleware = (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    return res.status(401).json({ message: 'Token não fornecido' });
  }
  try {
    const decoded = jwt.verify(token, SECRET_KEY);
    req.user = decoded;
    next();
  } catch (err) {
    return res.status(401).json({ message: 'Token inválido ou expirado' });
  }
};

// Middleware para controle de acesso
const adminMiddleware = (req, res, next) => {
  if (req.user.perfil !== 'admin') {
    return res.status(403).json({ message: 'Acesso negado' });
  }
  next();
};

// Endpoint de login
app.post('/api/auth/login', (req, res) => {
```

```
const { username, password } = req.body;
const user = users.find(u => u.username === username && u.password === password);
if (!user) {
  return res.status(401).json({ message: 'Credenciais inválidas' });
}
const token = jwt.sign({ id: user.id, perfil: user.perfil }, SECRET_KEY, { expiresIn:
'1h' });
res.json({ token });
});

// Endpoint para recuperação dos dados do usuário logado
app.get('/api/auth/me', defaultauthMiddleware, (req, res) => {
  const user = users.find(u => u.id === req.user.id);
  res.json({ user });
});

// Endpoint para recuperação de todos os usuários (somente admin)
app.get('/api/users', defaultauthMiddleware, adminMiddleware, (req, res) => {
  res.status(200).json({ data: users });
});

// Endpoint para recuperação de contratos (com prevenção de injeção)
app.get('/api/contracts', defaultauthMiddleware, adminMiddleware, (req, res) => {
  const { empresa, inicio } = req.query;
  if (!empresa || !inicio) {
    return res.status(400).json({ message: 'Parâmetros insuficientes' });
  }
  if (/^[^a-zA-Z0-9 _-]/.test(empresa) || /^[^0-9-]/.test(inicio)) {
    return res.status(400).json({ message: 'Parâmetros inválidos' });
  }
  const contracts = getContracts(empresa, inicio);
  res.status(200).json({ data: contracts });
});

// Mock de consulta ao banco (parametrizado)
function getContracts(empresa, inicio) {
  // Exemplo real deve usar prepared statements ou ORM
  return [{ empresa, inicio, contrato: 'Contrato Exemplo' }];
}

app.listen(port, () => {
  console.log(`Servidor rodando na porta ${port}`);
});
```

9. Utilizando um cliente (Insomnia, Postman ou outro de sua preferência), realize testes na API, garantindo que todos os pontos acima foram tratados.

### Exemplo 1: Login

Método: POST

URL: `http://localhost:3000/api/auth/login`

Cabeçalhos:

Body (JSON): json

```
{
  "username": "admin",
  "password": "123456789"
}
```

O retorno será:

The screenshot shows the Postman interface for a POST request to `http://localhost:3000/api/auth/login`. The request body is a JSON object with the following structure:

```
{
  "username": "admin",
  "password": "123456789"
}
```

The response is a 200 OK status with a response time of 151 ms and a body size of 411 B. The response body is a JSON object with a token:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTI0LCJwZXJmaWwiOiJhZG1pb2IiIm1hdCI6MTczNDE4ODQxNSwiZXhwIjozMTM0MTkyMDE1fQ.9babjBQ0bMhLjKpaNGQTF-HF22V6bJNuYbfQESY_VMg"
}
```

The Postman interface also shows the 'Body' tab selected, and the 'JSON' format chosen. The 'Send' button is visible, and the 'Save' button is also present. The 'Headers' tab is also visible, showing 7 headers. The 'Test Results' tab is also visible, showing a 200 OK status. The 'Cookies' tab is also visible, showing no cookies. The 'Headers' tab is also visible, showing 7 headers. The 'Test Results' tab is also visible, showing a 200 OK status. The 'Cookies' tab is also visible, showing no cookies.



## Exemplo 2: Recuperar dados do usuário logado

Método: GET

URL: `http://localhost:3000/api/auth/me`

Cabeçalhos:

```
Authorization: Bearer "token"
```

O retorno será:

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

- URL:** `http://localhost:3000/api/auth/me`
- Method:** GET
- Headers (10):**
  - Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTI0LCJwZXJmaWw0IjphZG1pb2I6ImhhdCI6MTczNDE4ODQxNSwiZXhwIjoxNzM0M0kyMDE1fQ.9babjBQ0bMhLjKpaNGQTF-HF22V5bJNuYbfQESY\_VMg
- Status:** 200 OK, 17 ms, 342 B
- Body (JSON):**

```
{  "user": {    "username": "admin",    "password": "123456789",    "id": 124,    "email": "admin@dominio.com",    "perfil": "admin"  }}
```

At the bottom of the interface, there is a Windows taskbar with the text "Ativar o Windows" and a notification "Acesse Configurações para ativar o Windows". The taskbar also shows icons for Postbot, Runner, Start Proxy, Cookies, Vault, Trash, and a help icon.

### Exemplo 3: Recuperar usuários (somente admin)

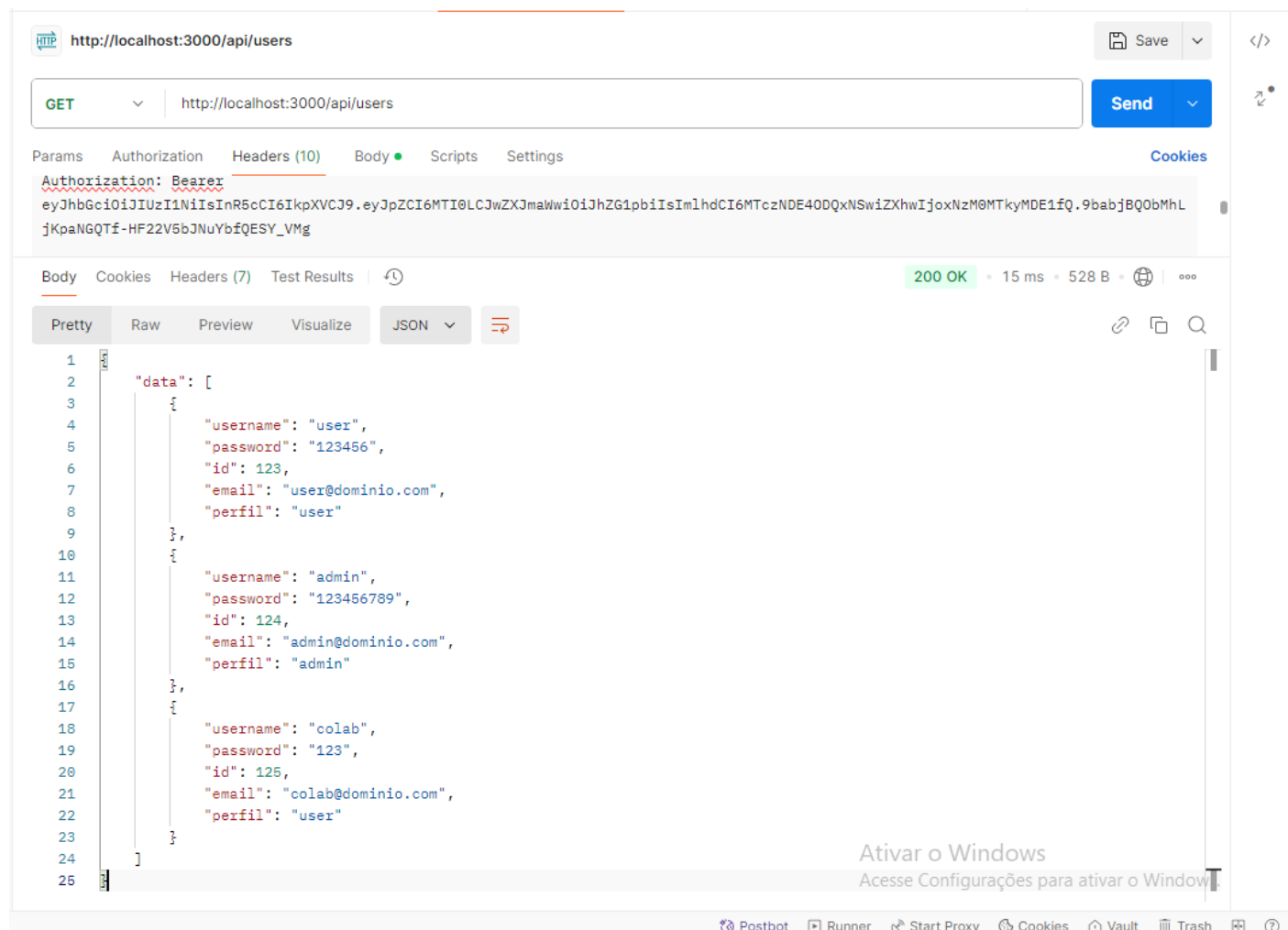
Método: GET

URL: `http://localhost:3000/api/users`

Cabeçalhos:

```
Authorization: Bearer "token"
```

O retorno será:



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

- URL:** `http://localhost:3000/api/users`
- Method:** GET
- Authorization Header:** `Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTI0LCJwZXJmaWwiOiJhZG1pb1IsIm1hdCI6MTczNDE4ODQxNSwiZXhwIjoxNzMTkyMDE1fQ.9babjBQ0bMhLjKpaNGQTf-HF22V5bJNuYbfQESY_VMg`
- Status:** 200 OK, 15 ms, 528 B
- Response Body (JSON):**

```
{  "data": [    {      "username": "user",      "password": "123456",      "id": 123,      "email": "user@dominio.com",      "perfil": "user"    },    {      "username": "admin",      "password": "123456789",      "id": 124,      "email": "admin@dominio.com",      "perfil": "admin"    },    {      "username": "colab",      "password": "123",      "id": 125,      "email": "colab@dominio.com",      "perfil": "user"    }  ]}
```

Ativar o Windows  
Acesse Configurações para ativar o Windows

## Exemplo 4: Recuperar contratos

Método: GET

URL: `http://localhost:3000/api/contracts?empresa= Software House&inicio=2024-01-01`

Cabeçalhos:

`Authorization: Bearer "token"`

O retorno será:

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

- URL:** `http://localhost:3000/api/contracts?empresa= Software House&inicio=2024-01-01`
- Method:** GET
- Headers:** `Authorization: Bearer`
- Status:** 200 OK, 73 ms, 326 B
- Body (JSON):**

```
1 {
2   "data": [
3     {
4       "empresa": "Software House",
5       "inicio": "2024-01-01",
6       "contrato": "Contrato Exemplo"
7     }
8   ]
9 }
```

At the bottom of the interface, there is a Windows watermark: "Ativar o Windows. Acesse Configurações para ativar o Windows."