

# Aula 6 – Segurança de aplicações web

---

PROFESSOR: HARLEY MACÊDO DE MELLO

# Roteiro

---

- Conceitos
- Criptografia de dados
- Recuperação de senha
- Sessão do usuário com JWT
- Recurso autorizado com CORS

# Conceitos

---

- Aumento de ataques aos sistemas
- LGPD garante vários direitos aos usuários
- Devido sua exposição na rede, sistemas web precisam ser muito seguros
- Várias técnicas podem ser usadas para melhorar a segurança
- Fatores humanos também podem comprometer os sistemas

# Criptografia de dados

---

- Forma de alterar um dado, usando uma fórmula de embaralhamento
- Uma senha mestre pode ser necessária para ser usada na fórmula
- Dados podem ser visualizados, mas não são compreensíveis
- Biblioteca 'bcrypt' trabalha com criptografia no NodeJS

# Criptografia de dados

---

```
68 //Rota para cadastrar usuário novo, criptografando a senha
69 app.post('/usuario/novo', (req, res) => {
70   try {
71     const {nome, senha} = req.body
72     var salt = bcrypt.genSaltSync(10)
73     var senhaParaSalvar = bcrypt.hashSync(senha, salt)
74     usuarios.push({nome: nome, senha: senhaParaSalvar})
75     res.json({mensagem: 'Usuário cadastrado com sucesso'})
76   } catch (error) {
77     res.json({mensagem: 'Erro durante a consulta', erro: error.message})
78   }
79 })
```

Criptografando a senha do usuário, para salvar no banco de dados.

# Criptografia de dados

---

```
81 //Simulação de um comparação de senha para um processo de login
82 app.post('/usuario/comparar', (req, res) => {
83   try {
84     const {nome, senha} = req.body
85     const usuario = usuarios.find((usuario) => {return usuario.nome === nome})
86     const resultado = bcrypt.compareSync(senha, usuario.senha)
87     res.json({resultado: resultado})
88   } catch (error) {
89     res.json({mensagem: 'Erro durante a consulta', erro: error.message})
90   }
91 })
```

Fazendo comparação  
de senha criptografada.

# Criptografia de dados

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

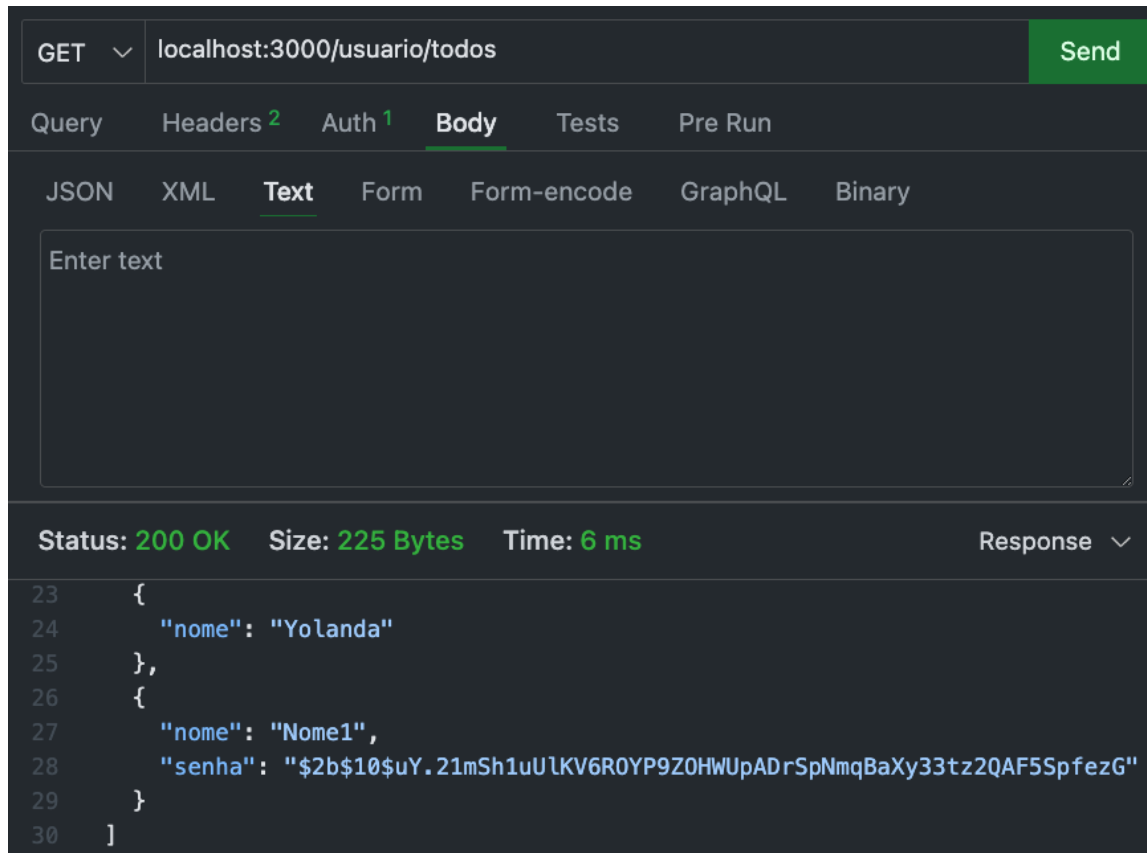
- Method:** POST
- URL:** localhost:3000/usuario/novo
- Body:** JSON format with the following content:

```
1 {  
2   "nome": "Nome1",  
3   "senha": "12345"  
4 }
```
- Status:** 200 OK
- Size:** 46 Bytes
- Time:** 36 ms
- Response:** JSON format with the following content:

```
1 {  
2   "mensagem": "Usuário cadastrado com sucesso"  
3 }
```

Fazendo request para criar novo usuário com nome e senha.

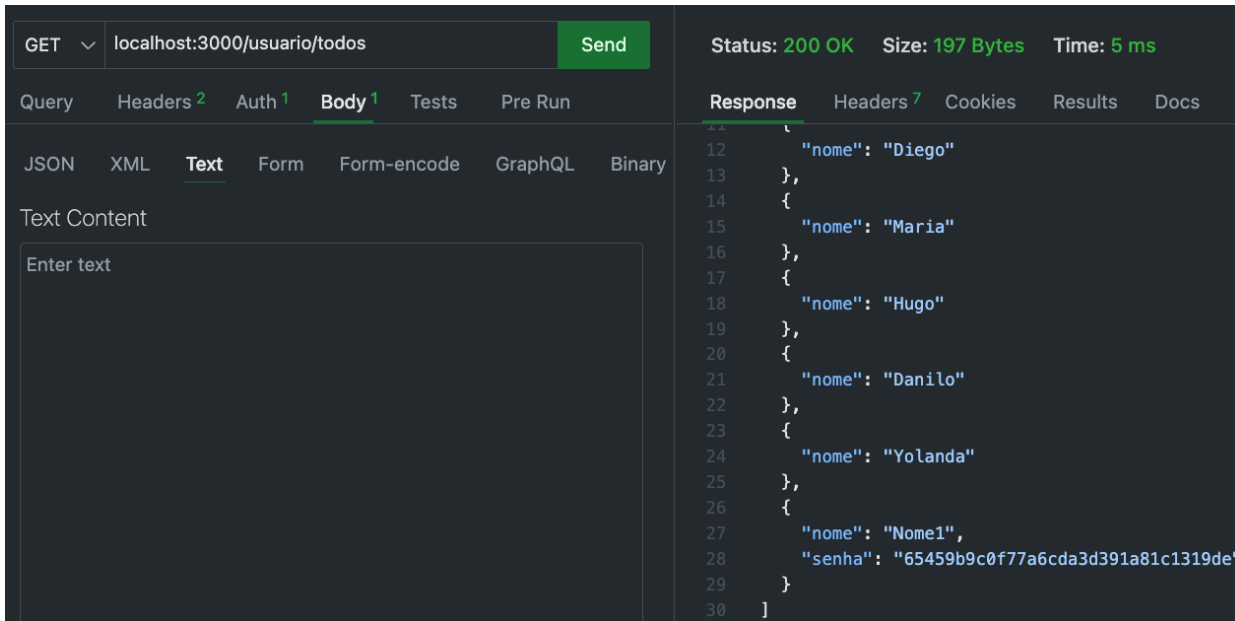
# Criptografia de dados



Verificando que o usuário foi cadastrado com a senha criptografada.



# Criptografia de dados



The screenshot shows a REST client interface. The top bar indicates a GET request to `localhost:3000/usuario/todos` with a status of `200 OK`, size of `197 Bytes`, and time of `5 ms`. The left sidebar shows the 'Body' tab selected, with a 'Text' content type. The main area displays the response body as a JSON array of user objects. The first five users are Diego, Maria, Hugo, Danilo, and Yolanda. The sixth user, 'Nome1', has a password field containing a long alphanumeric string, which is the encrypted password.

```
11 {
12   "nome": "Diego"
13 },
14 {
15   "nome": "Maria"
16 },
17 {
18   "nome": "Hugo"
19 },
20 {
21   "nome": "Danilo"
22 },
23 {
24   "nome": "Yolanda"
25 },
26 {
27   "nome": "Nome1",
28   "senha": "65459b9c0f77a6cda3d391a81c1319de"
29 }
30 ]
```

Resultado da criação do novo usuário, com a senha criptografada.

# Criptografia de dados

Fazendo request para criar novo usuário com nome e senha.

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

- Method:** POST
- URL:** localhost:3000/usuario/comparar
- Body:** JSON format with the following content:

```
1 {  
2   "nome": "Nome1",  
3   "senha": "12345"  
4 }
```
- Status:** 200 OK
- Size:** 18 Bytes
- Time:** 101 ms
- Response:** JSON format with the following content:

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

# Recuperação de senha

---

- Etapa crítica no sistema
- Para evitar fraude, um token deve ser enviado para o e-mail do usuário
- O token tem uma validade de alguns minutos, para evitar que seja usado no futuro
- O usuário deve cadastrar uma nova senha com regras de força
- Não se deve confiar apenas nas restrições definidas no Frontend
- O usuário mal intencionado pode burlar a formatação exigida nos formulários

# Recuperação de senha

---

```
75 app.get('/usuario/recuperarSenha', (req, res) => {  
76   try {  
77     //Gerar token  
78     let tokenRecuperarSenha = crypto.randomBytes(10).toString('hex')  
79     //Enviar email com token  
80     //Atualizar BD com redefinição de senha  
81     res.json({codigo: tokenRecuperarSenha})  
82   } catch (error) {  
83     res.json({mensagem: 'Erro durante a consulta'})  
84   }  
85 })
```

Token para recuperação de senha sendo gerado.

# Recuperação de senha

GET

localhost:3000/usuario/recuperarSenha

Query

Headers 2

Auth 1

Body 1

Tests

Pre Run

JSON

XML

Text

Form

Form-encode

GraphQL

Binary

Form Fields

Status: 200 OK   Size: 33 Bytes   Time: 32 ms

1

{

2

"codigo": "820ca7a77e34b0771d71"

3

}

Exemplo de token gerado, porém o mesmo seria enviado para o e-mail do usuário.

# Sessão do usuário com JWT

---

- Usuário precisa manter uma sessão de navegação segura
- JWT Envolve a autenticação e autorização através de token
- Faz-se login para obter um token
- Então cada requisição precisa utilizar o token recebido
- O token pode ter uma validade
- Instalar jwt com 'npm Install jsonwebtoken'
- Importar jwt com 'require("jsonwebtoken")'

# Sessão do usuário com JWT

---

```
37 //Rota para validar login com usuário e senha
38 app.post('/validaLogin', (req, res) => {
39   try {
40     const {usuario, senha} = req.body
41     if (usuario === process.env.USUARIO && senha === process.env.SENHA) {
42       let novoToken = jwt.sign({usuario}, process.env.APP_KEY, {expiresIn: 9000})
43       res.json({logado: true, token: novoToken})
44     } else {
45       res.json({logado: false, mensagem: 'Usuário ou senha errados.'})
46     }
47   } catch (error) {
48     res.json({logado: false, mensagem: 'Erro durante o login.'})
49   }
50 })
51
```

Rota que valida usuário e senha e cria um token para acessos subsequentes.

# Sessão do usuário com JWT

---

```
14 //Verificar se a requisição possui token válido, e portanto, o usuário está logado
15 const verificarJWT = (req, res, next) => {
16     const token = req.body.token
17     if (!token) {
18         res.json({logado: false, mensagem: 'Token não foi enviado.'})
19     }
20     jwt.verify(token, process.env.APP_KEY, (err, decoded) => {
21         if (err) {
22             res.json({logado: false, mensagem: 'Falha na autenticação'})
23         }
24     })
25     next()
26 }
```

Rota que verifica a existência de um token válido e permite a navegação.



# Sessão do usuário com JWT

---

```
52 //Rota para obter usuários, com necessidade de informar token
53 app.get('/usuario/todos', verificarJWT, (req, res) => {
54     try {
55         res.json(usuario)
56     } catch (error) {
57         res.json({mensagem: 'Erro durante a consulta'})
58     }
59 })
```

Rota que requer a passagem de um token para ser concluída.

# Recurso autorizado com CORS

---

- Cross Origin Resource Sharing
- Algumas APIs podem ser definidas para uso próprio
- A origem do Frontend deve ser conhecida
- Através do domínio pode-se definir quem tem acesso aos recursos
- Instalar com 'npm install cors'
- Habilitar com 'app.use(cors())'

# Recurso autorizado com CORS

---

```
14 //Configurando o CORS
15 app.use(cors({
16   origin: ['https://node-express-api-rest-mock.vercel.app',
17           'https://node-express-api-rest-mock.herokuapp.com']
18 })
19 }
```

Habilitando o CORS e passando as origens permitidas.