Sessões e Criptografia em Desenvolvimento Web

Nesta aula, vamos explorar conceitos fundamentais de segurança e gerenciamento de estado em aplicações web. Você aprenderá como manter usuários conectados de forma segura e proteger informações sensíveis.

O Protocolo HTTP é Sem Estado

O Desafio

O protocolo HTTP é **stateless**, ou seja, cada requisição é independente. O servidor não "lembra" automaticamente quem fez uma requisição anterior.

Imagine entrar em uma loja onde o vendedor esquece quem você é cada vez que você muda de corredor. Frustrante, não é?

A Solução

Para resolver esse problema, precisamos de um mecanismo que permita ao servidor **reconhecer o usuário** entre diferentes requisições.

É aí que entram as **sessões** – uma forma inteligente de manter a "memória" do servidor sobre cada usuário conectado.

O Que É uma Sessão?

Uma **sessão** é um mecanismo que permite ao servidor **manter dados do usuário entre requisições HTTP**. Quando você faz login em um site e continua navegando sem precisar fazer login novamente, isso é graças às sessões.

As sessões funcionam como uma "gaveta virtual" no servidor, onde informações sobre você são temporariamente armazenadas. O servidor identifica qual gaveta é sua através de um pequeno código especial chamado **session ID**.

Como Funciona uma Sessão?

Usuário faz login

O servidor valida as credenciais do usuário

Servidor cria sessão

Gera um ID único e armazena dados no servidor

Envia cookie

O navegador recebe um cookie com o session ID

Requisições seguintes

O navegador envia o cookie em cada requisição

O Que Podemos Armazenar em Sessões?

Identificação do Usuário

- ID do usuário logado
- Nome e email
- Permissões de acesso

Preferências Temporárias

- Idioma selecionado
- Configurações de interface
- Filtros aplicados

Dados de Navegação

- Carrinho de compras
- Histórico de navegação
- Formulários parcialmente preenchidos

Todos esses dados ficam **armazenados no servidor**, garantindo maior segurança e controle sobre informações sensíveis dos usuários.

Sessões vs. localStorage

Sessões (req.session)

- Dados armazenados **no servidor**
- Mais seguro para informações sensíveis
- Controlado pelo desenvolvedor
- Expira quando o usuário faz logout

localStorage

- Dados armazenados **no navegador**
- Qualquer pessoa pode ver e modificar
- Persiste mesmo após fechar o navegador
- Útil apenas para preferências visuais

Onde os Dados Ficam Armazenados?

No Servidor

As informações da sessão (`req.session`) ficam **armazenadas no servidor**, geralmente em memória ou em um banco de dados dedicado como Redis.

Isso garante que informações sensíveis, como IDs de usuário e permissões, não possam ser acessadas ou modificadas diretamente pelo cliente.

No Cliente

O navegador do usuário armazena apenas um pequeno **identificador da sessão**, geralmente através de um cookie.

Este identificador funciona como uma "chave" que o navegador apresenta ao servidor em cada requisição, permitindo que o servidor recupere os dados corretos da sessão.

Como fazer uma sessão?

```
<!DOCTYPE html>
<html>
<head><title>Exemplo Sessão</title></head>
<body>
  <% if (nome) { %>
    <h1>Olá, <%= nome %>!</h1>
    <a href="/logout">Sair</a>
  <% } else { %>
    <h1>Você não está logado.</h1>
    <a href="/login">Entrar</a>
  <% } %>
</body>
</html>
```

Como fazer uma sessão?

```
const session = require('express-session');

const express = require('express');
const session = require('express-session');
const app = express();

app.set('view engine', 'ejs');

// Configurando o uso de sessões
app.use(session({
  secret: 'chave-secreta-bem-dificil', // usada para assinar o ID da sessão
  resave: false,
  saveUninitialized: true,
  cookie: { maxAge: 60000 } // duração da sessão (1 min)
}));

// Login (simples)
app.get('/login', (req, res) => {
  req.session.nome = 'Luís'; // simulando login
  res.redirect('/');
});

// Logout
app.get('/logout', (req, res) => {
  req.session.destroy((err) => {
    res.redirect('/');
  });
});
```

Destruindo uma Sessão

Quando um usuário faz **logout**, precisamos destruir a sessão para garantir que ele não continue autenticado. Ao destruir a sessão:

- Os dados armazenados no servidor são removidos
- O cookie com o session ID perde validade
- O "login some" e o usuário precisa se autenticar novamente

Isso é fundamental para a **segurança da aplicação**, especialmente em computadores compartilhados ou públicos.

⚠️ Cuidado com o localStorage

> **Nunca armazene senhas, tokens de autenticação ou informações sensíveis no localStorage.**

O localStorage é visível e editável por qualquer pessoa com acesso ao navegador. Qualquer script malicioso pode ler ou modificar esses dados, comprometendo a segurança do usuário.

Use localStorage apenas para configurações que não comprometem a segurança, como:

- Preferências de tema (modo escuro/claro)
- Configurações de interface do usuário
- Dados não sensíveis que melhoram a experiência

Criptografia de Senhas

Armazenar senhas em **texto puro** no banco de dados é uma das piores práticas de segurança. Se o banco for comprometido, todas as senhas ficam expostas.

A solução é usar **criptografia** para transformar a senha em um código irreversível antes de salvá-la. Mesmo que alguém tenha acesso ao banco de dados, não conseguirá descobrir a senha original.

Apresentando o bcrypt

	Signatures			KEM		Encryption		KEM/ Encryption		Overall	
		NIST	F.		F.		F.	NIST	F.	NIST	F.
Lattice-based	CRYSTALS-DILITHIUM DRS FALCON pqNTRUSign qTESLA	4	5	CRYSTALS-KYBER Ding Key Exchange FrodoKEM HILA5 KINDI LAC LIMA Lizard LOTUS NewHope NTRUEncrypt NTRU-HRSS-KEM NTRU Prime OKCN/AKCN/CNKE Round2 SABER Three Bears Titanium	18	Compact LWE EMBLEM and R.EMBLEM Giophantus KINDI LAC LIMA Lizard LOTUS NTRUEncrypt Odd Manhattan OKCN/AKCN/CNKE Round2 Titanium	13	24	31	28	36
Code-based	pqsigRM RaCoSS RankSign	5	3	BIG QUAKE BIKE Classic McEliece DAGS Edon-K HQC LAKE LEDAkem LOCKER NTS-KEM Ouroboros-R QC-MDPC-KEM Ramstake RLCE-KEM RQC	15	LEDApkc McNie	2	19	17	24	20
Multi-variate	DualModeMS GeMSS Gui HiMQ-3 MQDSS LUOV Rainbow SRTPI	7	8	CFPKM DME	2	SRTPI	1	6	3	13	11
Hash-based	Gravity-SPHINCS SPHINCS+	4	2							4	2
Braids	WalnutDSA	1	1					1	0	2	1
Chebychev				RVB	1			1	1	1	1
Finite Automata	Picnic	1	1					1	0	2	1
Hypercomplex Numbers				HK17	1			1	1	1	1
Isogeny				SIKE	1			1	1	1	1
LPN				Lepton	1			0	1	0	1
Mersenne Numbers				Mersenne-756839	1			0	1	0	1
Rand. Walk						Guess Again	1	1	1	1	1
RSA	Post-quantum RSA-Signature	1	1	Post-quantum RSA-Encryption	1	Post-quantum RSA-Encryption	1	1	2	2	3
?								1	0	1	0
Total		23	21		41		18	57	59	80	80

O Que É?

bcrypt é uma biblioteca de criptografia projetada especificamente para proteger senhas. Ela é amplamente usada e considerada uma das mais seguras para esse propósito.

Por Que Usar?

- Cria hashes **irreversíveis**
- Inclui proteção contra ataques de força bruta
- Adiciona "salt" automático para maior segurança

As Duas Funções Principais do bcrypt

bcrypt.hash()

Transforma a senha em um **hash irreversível** antes de salvar no banco de dados.

```
const hash = await bcrypt.hash(senha, 10);
```

O número 10 é o **saltRounds**, que define o nível de complexidade da criptografia.

bcrypt.compare()

Verifica se a senha digitada pelo usuário corresponde ao hash armazenado.

```
const valida = await bcrypt.compare(senha, hash);
```

Retorna `true` se a senha está correta ou `false` caso contrário.

Como Funciona o bcrypt.hash()?

01

Recebe a senha

A senha em texto puro é fornecida como primeiro parâmetro

02

Aplica o salt

Um valor aleatório único é gerado e misturado à senha

03

Processa com algoritmo

O bcrypt executa múltiplas rodadas de criptografia

04

Retorna o hash

Uma string longa e irreversível é retornada para salvar no banco

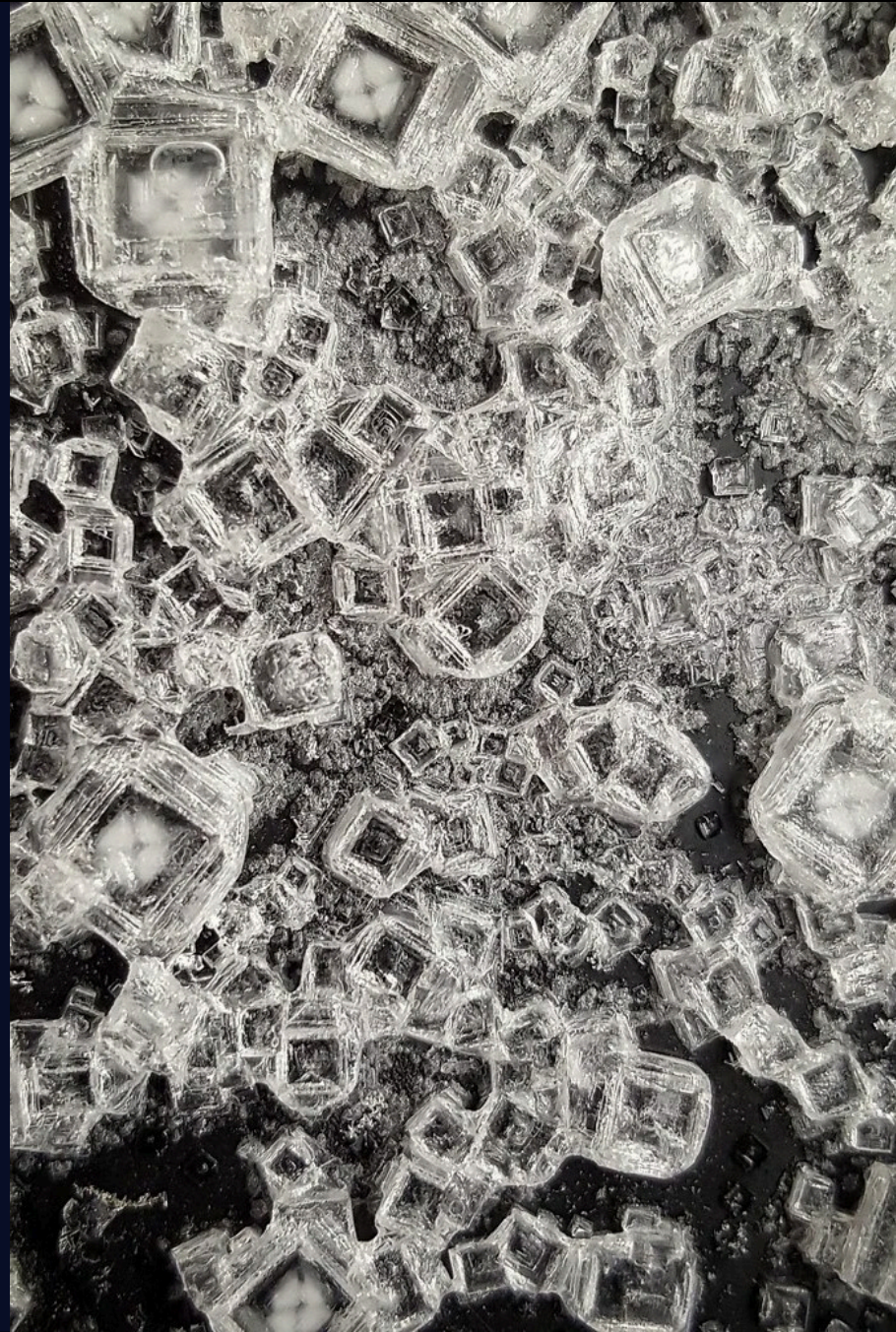
O **saltRounds** define quantas vezes o algoritmo processa a senha. Quanto maior o número, mais seguro (e mais lento) será o processo. O valor 10 é um bom equilíbrio entre segurança e desempenho.

O Poder do Salt

Um dos recursos mais importantes do bcrypt é o **salt** – um valor aleatório adicionado à senha antes da criptografia.

> Mesmo que duas pessoas usem a **mesma senha**, os hashes gerados serão **completamente diferentes**.

Isso protege contra ataques de "rainbow tables", onde hackers usam listas pré-calculadas de hashes comuns. Com o salt, cada senha gera um hash único, tornando esses ataques ineficazes.



Exemplo Prático: Cadastro de Usuário

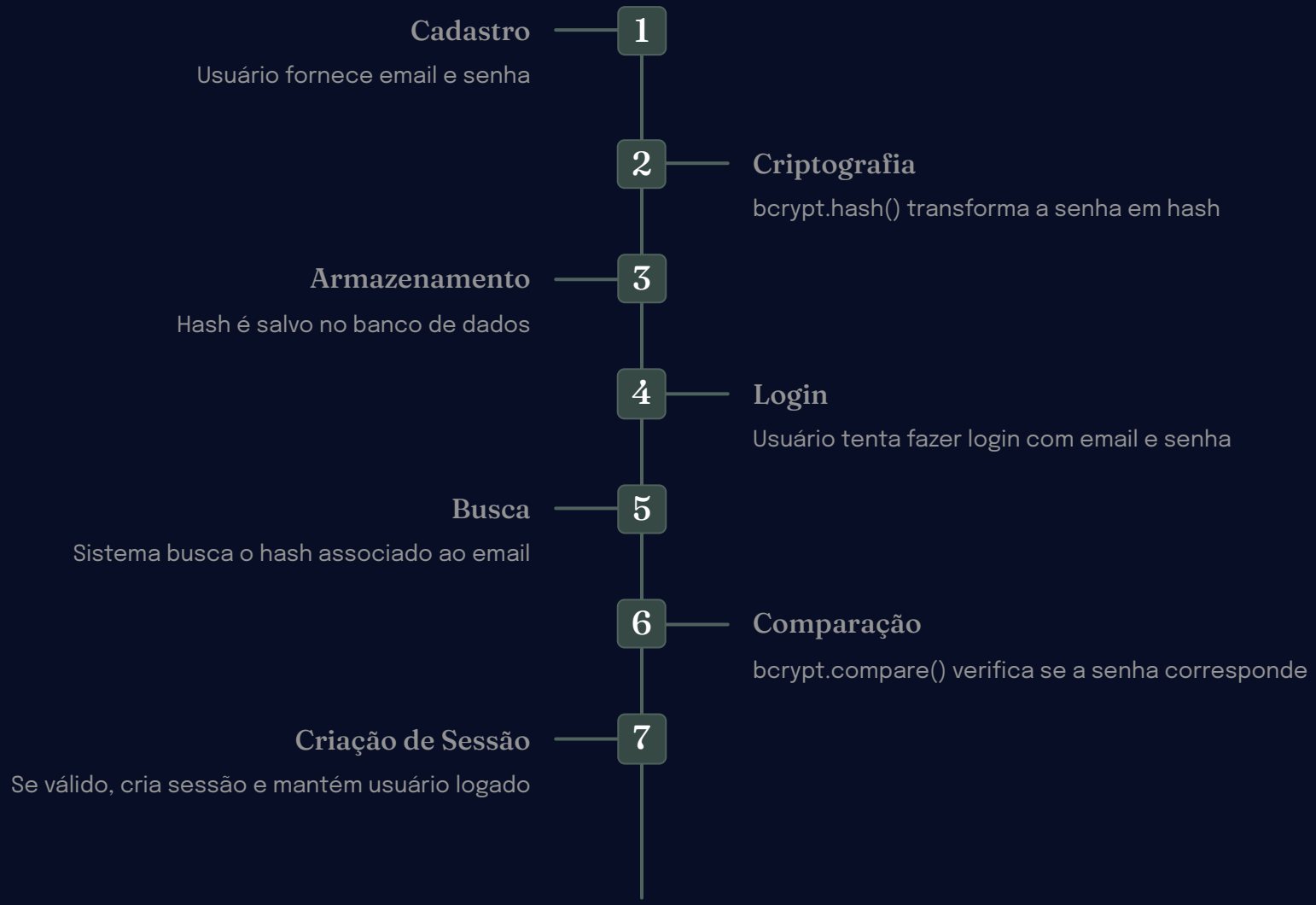
```
const bcrypt = require('bcrypt');

// Quando o usuário se cadastra
app.post('/cadastro', async (req, res) => {
  const { email, senha } = req.body;

  // Criptografa a senha antes de salvar
  const senhaCriptografada = await bcrypt.hash(senha, 10);

  // Salva no banco de dados
  await usuariosmodel.inserir({
    email: email,
    senha: senhaCriptografada
  });

  res.send('Usuário cadastrado com sucesso!');
});
```

Observe que **nunca salvamos a senha original** no banco. Apenas o hash gerado pelo bcrypt é armazenado.

Exemplo Prático: Login de Usuário

```
// Quando o usuário faz login
app.post('/login', async (req, res) => {
 const { email, senha } = req.body;

 // Busca o usuário no banco
 const usuario = await db.usuarios.buscarPorEmail(email);

 if (!usuario) {
 return res.status(401).send('Usuário não encontrado');
 }

 // Compara a senha digitada com o hash do banco
 const senhaValida = await bcrypt.compare(senha, usuario.senha);

 if (senhaValida) {
 req.session.userId = usuario.id;
 res.send('Login realizado com sucesso!');
 } else {
 res.status(401).send('Senha incorreta');
 }
});
```

Fluxo Completo: Do Cadastro ao Login



Boas Práticas de Segurança

✓ Sempre use bcrypt

Nunca armazene senhas em texto puro ou use métodos fracos como MD5

✓ Use sessões para autenticação

Mantenha dados sensíveis no servidor, não no cliente

✓ Destrua sessões no logout

Sempre limpe a sessão quando o usuário sair

✗ Evite localStorage para dados sensíveis

Use apenas para preferências visuais não-críticas

✓ Use HTTPS

Proteja a comunicação entre cliente e servidor

✓ Valide entradas

Sempre valide e sanitize dados recebidos do cliente

Resumo dos Conceitos

Sessões

- Mantêm dados do usuário entre requisições
- Armazenam informações **no servidor**
- Usam cookies para identificação
- Devem ser destruídas no logout

localStorage

- Armazena dados **no navegador**
- Não é seguro para informações sensíveis
- Útil apenas para preferências visuais

bcrypt

- Criptografa senhas de forma irreversível
- `bcrypt.hash()` → cria o hash
- `bcrypt.compare()` → valida a senha
- Adiciona salt automático para segurança extra

Segurança

- Nunca armazene senhas em texto puro
- Use HTTPS em produção
- Valide todas as entradas do usuário