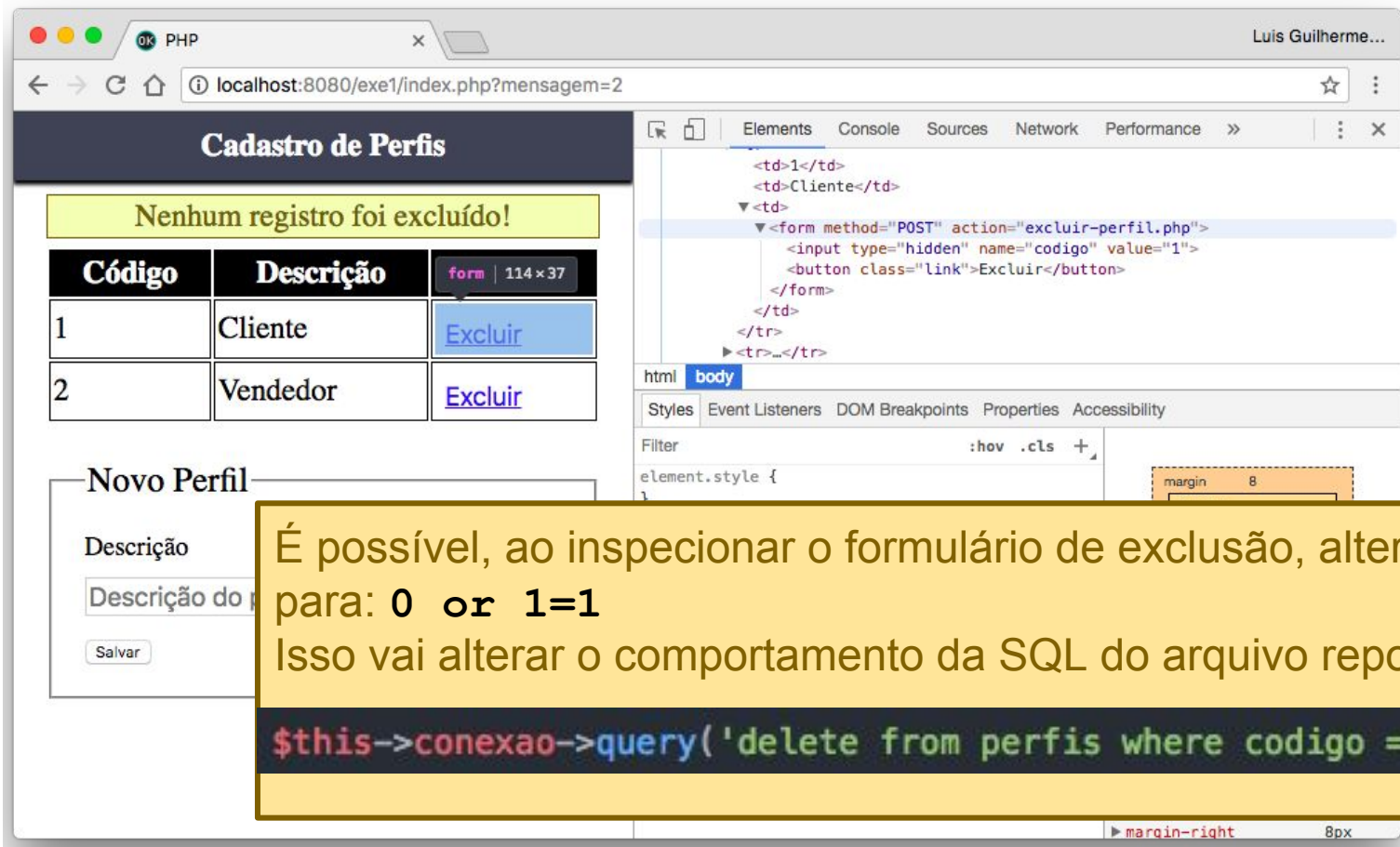


# PHP

## Aula 2

# Segurança no PHP

# Ataque SQL Injection (exe1/index.html)



**Cadastro de Perfis**

Nenhum registro foi excluído!

Código	Descrição	
1	Cliente	<a href="#">Excluir</a>
2	Vendedor	<a href="#">Excluir</a>

**Novo Perfil**

Descrição  
Descrição do perfil

Salvar

Elements

```
<td>1</td>
<td>Cliente</td>
<td>
  <form method="POST" action="excluir-perfil.php">
    <input type="hidden" name="codigo" value="1">
    <button class="link">Excluir</button>
  </form>
</td>
</tr>
<tr>
  <td>2</td>
  <td>Vendedor</td>
  <td>
    <form method="POST" action="excluir-perfil.php">
      <input type="hidden" name="codigo" value="1">
      <button class="link">Excluir</button>
    </form>
  </td>
</tr>
</tbody>
</table>
```

html body

Styles

Filter

element.style {

margin 8

É possível, ao inspecionar o formulário de exclusão, alterar o *value* para: **0 or 1=1**  
Isso vai alterar o comportamento da SQL do arquivo repositório.php

```
$this->conexao->query('delete from perfis where codigo = ' . $codigo);
```

margin-right 8px

# Ataque SQL Injection (exe1/index.html)

The screenshot shows a web browser window with the address bar displaying `localhost:8080/exe1/index.php?mensagem=3`. The page title is "Cadastro de Perfis". A yellow message box at the top states "Registro excluído com sucesso!". Below this is a table with three columns: "Código", "Descrição", and "Opções". Under the table, there is a section titled "Novo Perfil" containing a form with a label "Descrição", a text input field with the placeholder "Descrição do perfil", and a "Salvar" button.

Este ataque fará com que o código executado no banco de dados seja:

```
delete from perfis where codigo=0 or 1=1
```

Este comando excluirá TODOS os perfis do banco de dados.

# Proteção contra SQL Injection (exe1/index.php)

**Mas como proteger?** Ao invés de enviar a variável diretamente para SQL como fizemos no método `excluir`, devemos utilizar uma sintaxe alternativa:

```
$stmt = $this->conexao->prepare("delete from perfis where codigo = ?");  
$stmt->bind_param("i", $codigo);  
$stmt->execute();
```

Caractere	Descrição
i	corresponde a uma variável de tipo inteiro
d	corresponde a uma variável de tipo double
s	corresponde a uma variável de tipo string
b	corresponde a uma variável que contém dados para um blob e enviará em pacotes

# Proteção contra SQL Injection (exe2/index.php)

O método *prepare()* também pode ser utilizado em consultas. Note que o *bind\_param()* pode receber vários parâmetros, cada um respectivamente representado pelas letras do primeiro parâmetro. Note também que para consultas, devemos usar o método *get\_result()* para obter os dados.

```
$conexao = new mysqli('127.0.0.1','roupas_user','123eja','roupas_database');  
  
$email = $_POST['email'];  
$senha = $_POST['senha'];  
  
$query = $conexao->prepare('select nome from usuarios where email = ? and senha = ?');  
$query->bind_param("ss",$email,hash('sha256', $senha));  
$query->execute();  
$resultado = $query->get_result();
```

## Outras dicas gerais sobre segurança

- ✓ Nunca permita o tráfego de dados em claro (sem HTTPS)
- ✓ Não transmita dados sensíveis pela URL (senhas, por exemplo)
- ✓ Não mantenha dados sensíveis em cookies do navegador
- ✓ Nunca salve as senhas em claro no banco de dados. Procure utilizar funções hash criptográficas sem colisões conhecidas. Em outras palavras, não use MD5 ou SHA-1. Prefira: SHA256 ou SHA512.

Veja: [https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)

# Cookies & Sessions

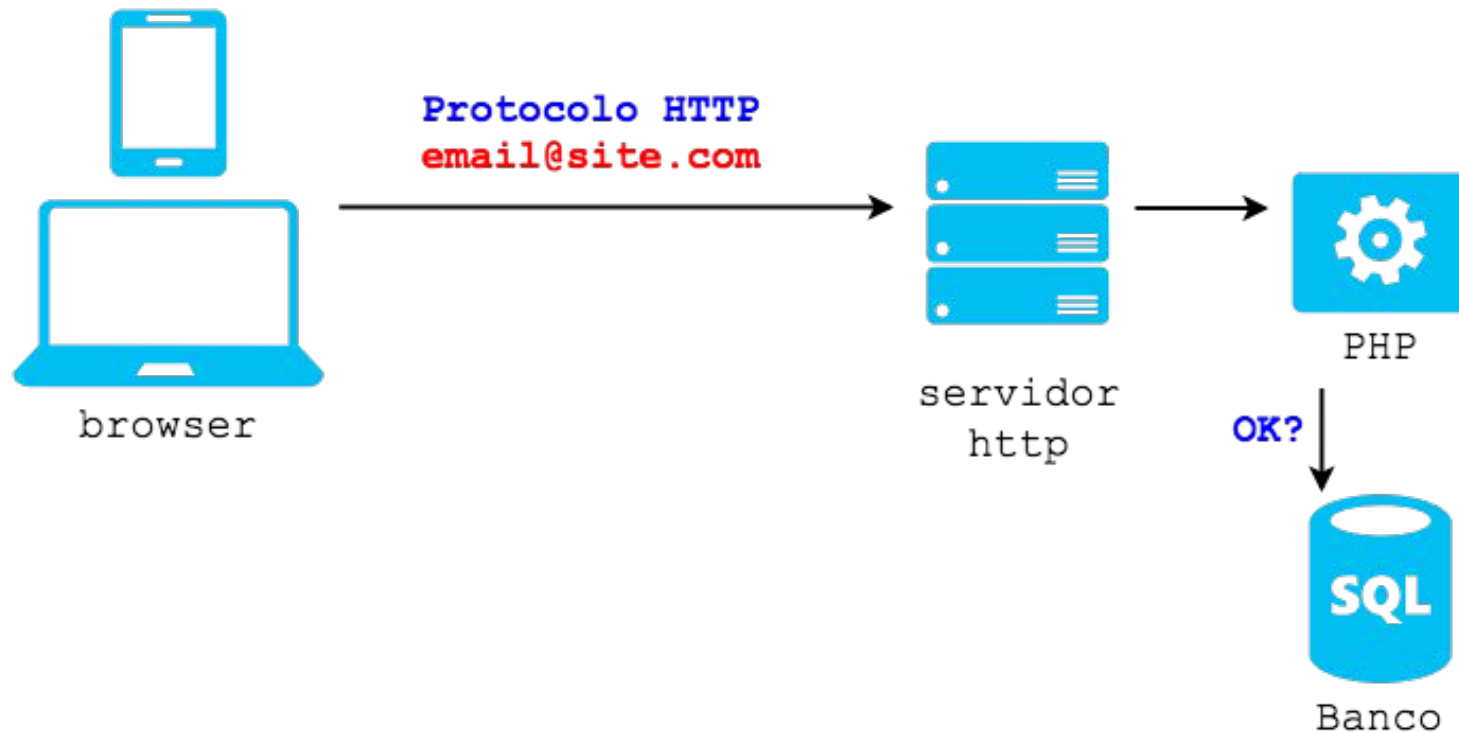
## Simulando uma página de login



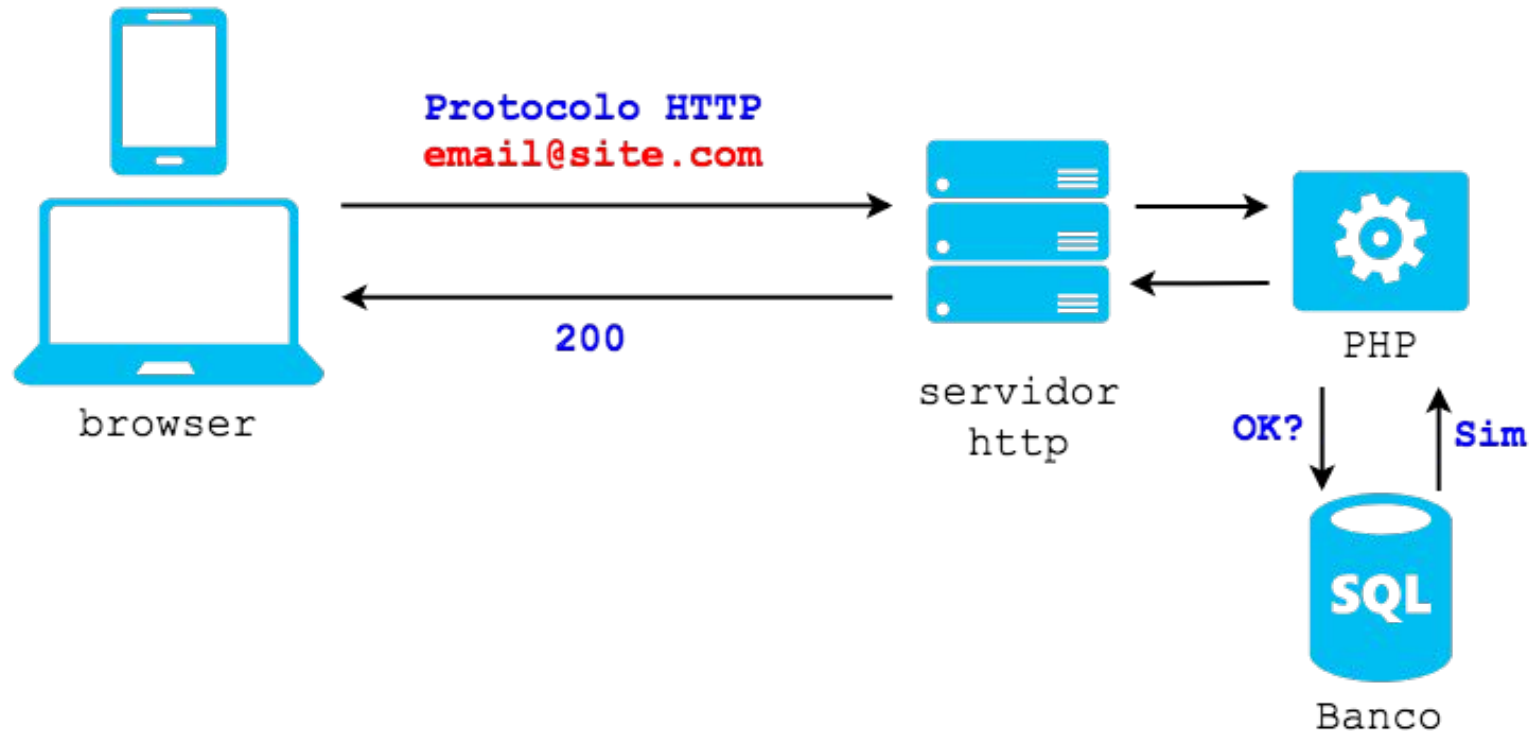
Primeiramente, o usuário envia suas informações ao servidor via HTTP (que é um protocolo stateless).



O servidor, através de um script PHP, verifica no banco de dados se as informações de login são válidas.



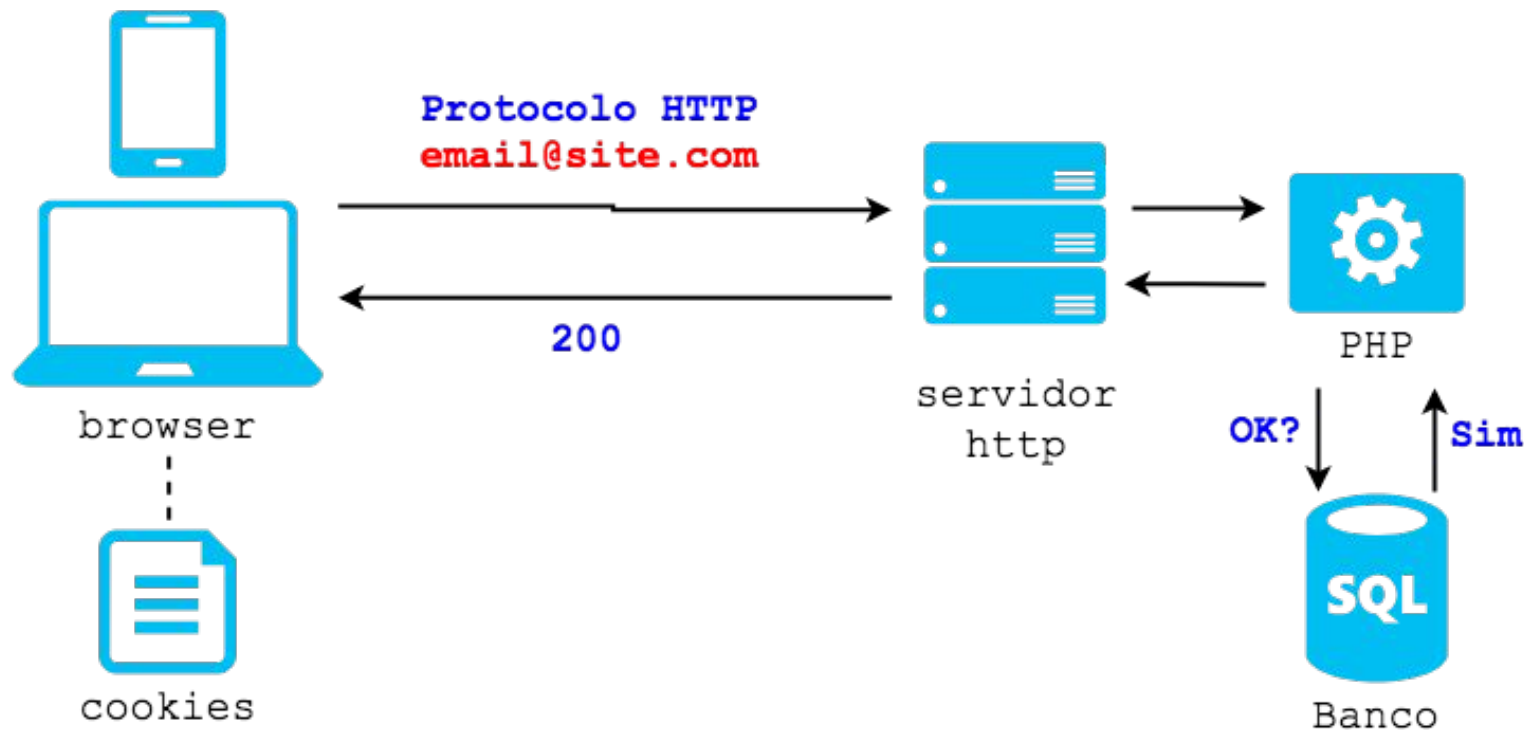
Supondo que as informações sejam válidas, o PHP retorna ao servidor, que por sua vez retorna ao usuário uma mensagem de sucesso.



O problema é que nem o browser, nem o servidor guardaram as informações de sucesso no login. Neste caso, o usuário teria que enviar novamente seus dados de login para o servidor validar (de novo).



O cliente pode guardar esses dados em um *cookie*! Um *cookie* é uma variável na qual o browser pode armazenar informações enviadas pelo servidor com intuito de reenviá-las na próxima solicitação.



É possível definir um cookie com a função `setcookie()`

```
$linha = $resultado->fetch_assoc();  
  
// Define um Cookie. Neste caso, ele vai durar até o usuário fechar o browser  
setcookie('usuario',$linha['nome']);  
  
header('Location: index.php?success=true');  
die();
```

Para acessá-lo, devemos usar `$_COOKIE`, informando como parâmetro do array o nome do cookie.

```
<?php if ( $_GET['success'] == true ): ?>  
    <span class="success">  
        Seja bem-vindo <?= $_COOKIE['usuario'] ?>!  
    </span>  
<?php endif; ?>
```

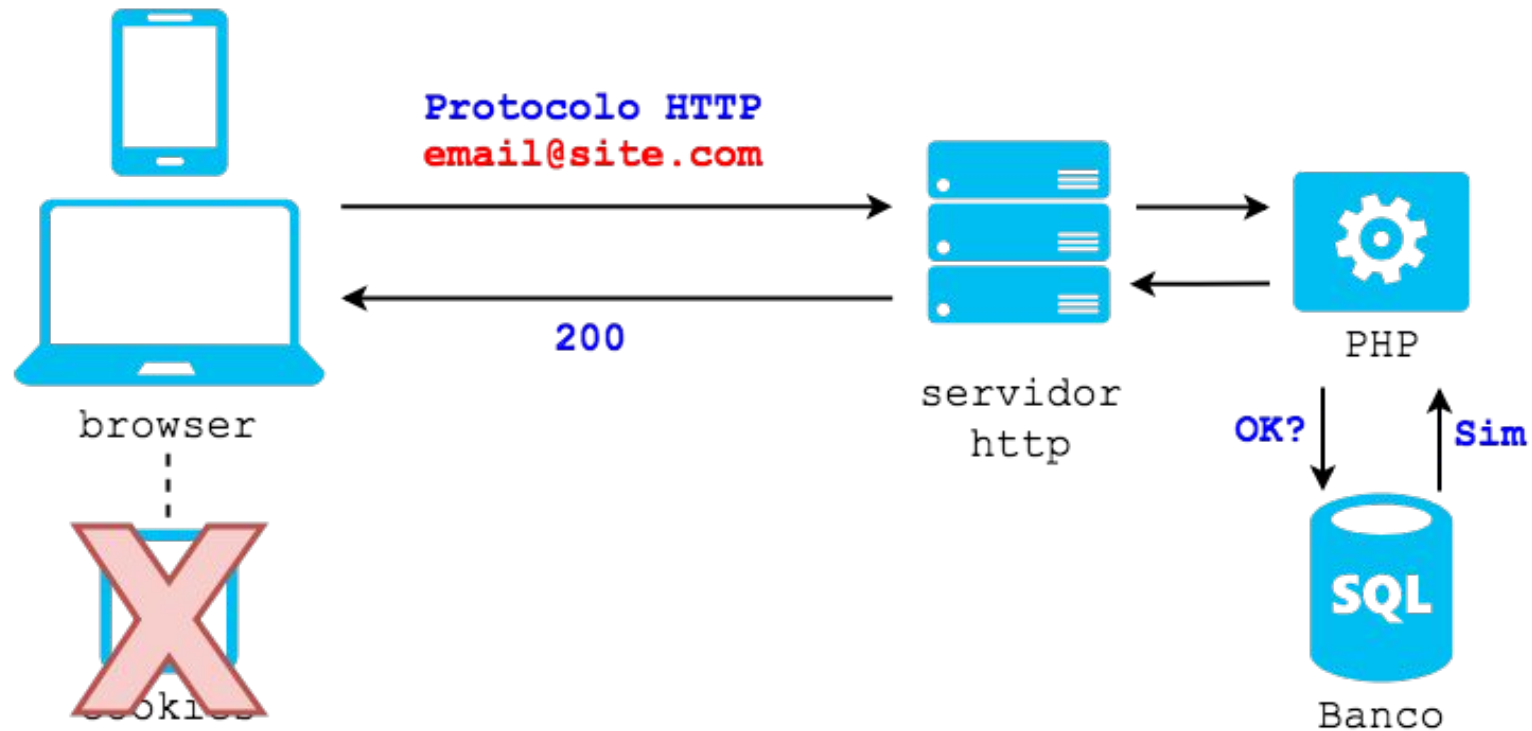


O problema, porém, é que como o *cookie* fica armazenado no browser, o usuário pode alterá-lo acessando a aba *Application* do *DevTools*. Neste exemplo, ele pode se passar por outro usuário!

The screenshot shows a web browser window with a login page. The page has a green header with the text "Seja bem-vindo Fulano!". Below the header, there is a login form with two input fields: "E-Mail" containing "email@site.com" and "Senha" (Password) containing masked characters. A "Login" button is positioned to the right of the password field. The browser's address bar shows the URL "127.0.0.1:8080/exe2/index.php?success=true". The Chrome DevTools "Application" tab is open, displaying the "Storage" section. Under "Storage", the "Cookies" folder is expanded, showing a cookie for "http://127.0.0.1:8080". The cookie table has columns for Name, Value, and expiration. The first row shows a cookie named "usuario" with a value of "Fulano" and an expiration of "13".

Name	Value	...	...	...	...	...	...	...	...
usuario	Fulano	...	...	...	13				

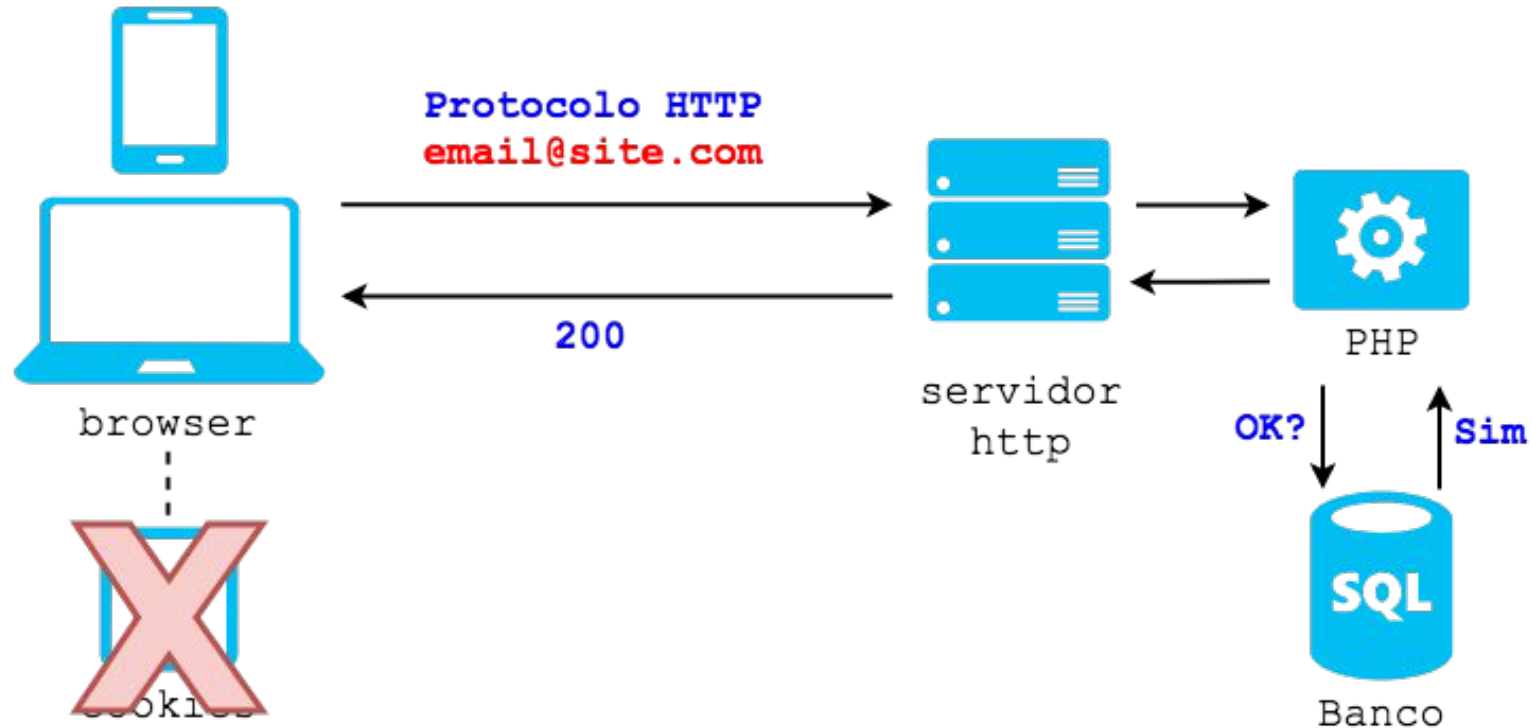
Então, **NUNCA** salve dados pelos quais o usuário possa tentar burlar a segurança de seu site.





Então, **NUNCA** salve dados pelos quais o usuário possa tentar burlar a segurança de seu site.

## Mas então onde vamos guardar os dados?

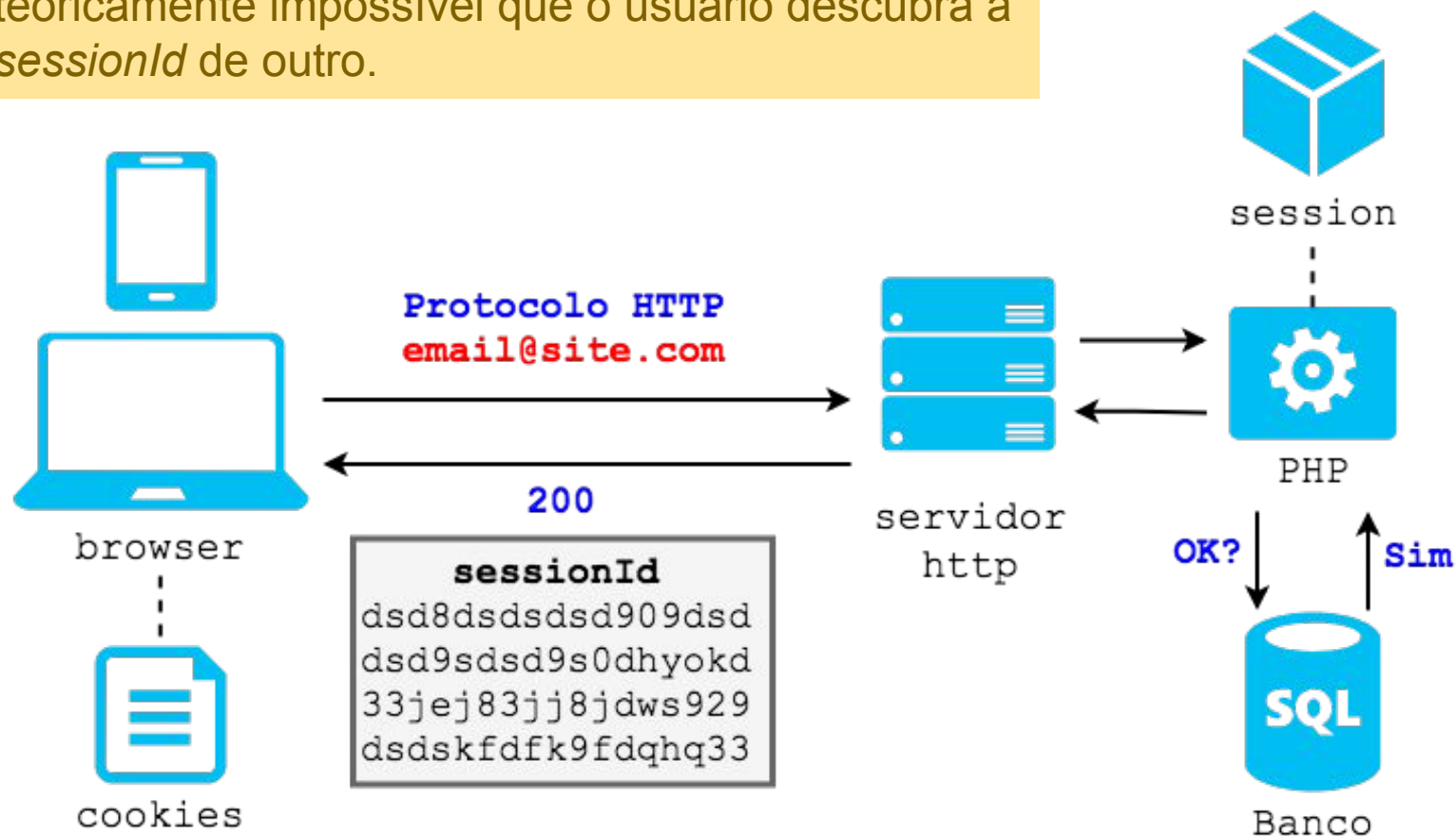


**No servidor!** Uma *session* é um "espaço" no servidor em que podemos guardar dados dos usuários. O servidor pode ter várias *sessions*, para armazenar vários dados. Cada *session* possui um ID de identificação.

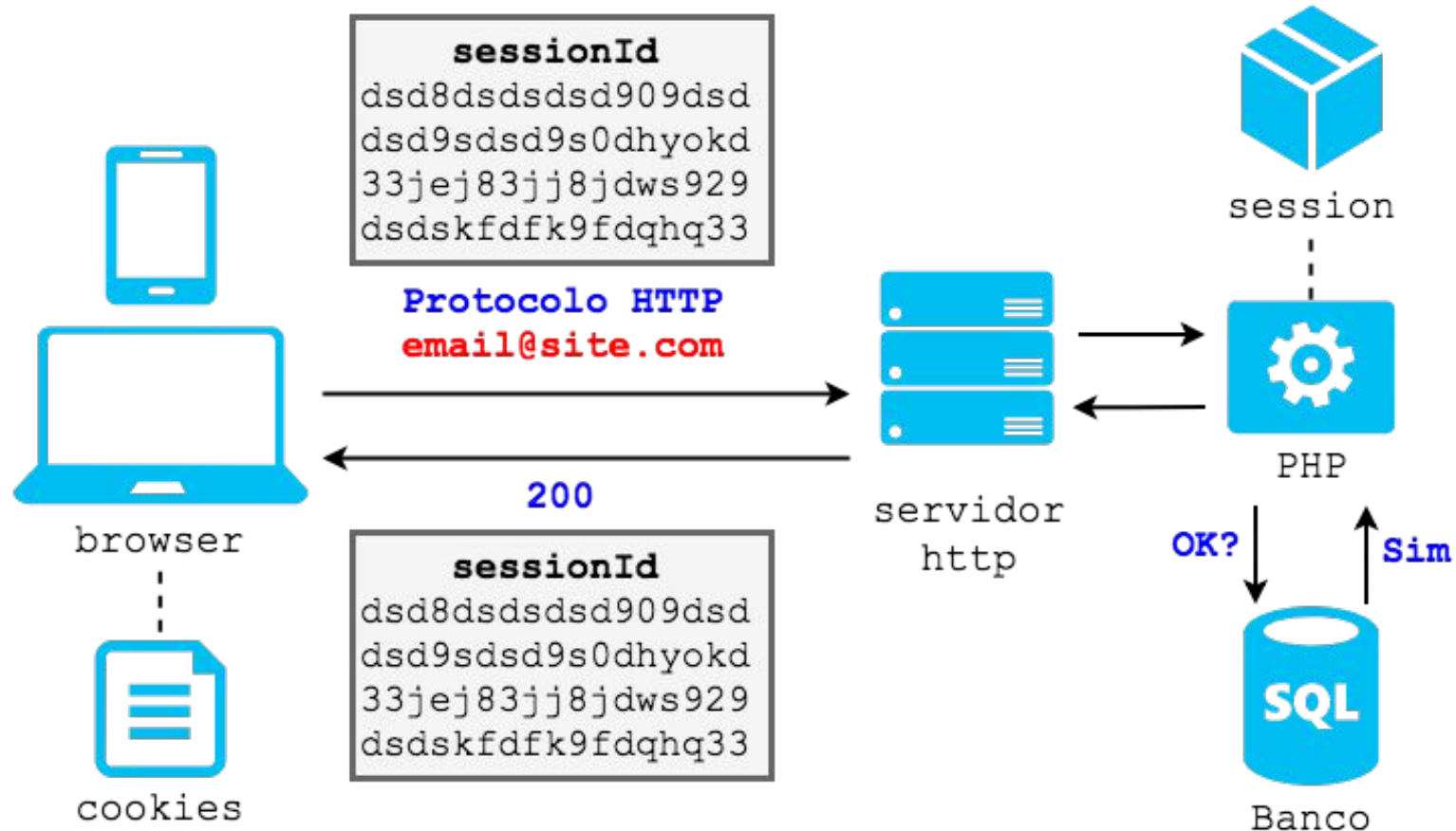


**Mas como o usuário vai saber em qual session estão os seus dados?**

**Simples**, o servidor salva o *sessionId* nos *cookies* do usuário. Como o *sessionId* é uma grande string aleatória e possui uma data limite, é teoricamente impossível que o usuário descubra a *sessionId* de outro.



Agora, a cada nova requisição o browser automaticamente envia a *sessionId* para o servidor recuperar a *session* associada ao usuário.



# Trabalhando com sessions (exe3/index.php)

**Passo 1:** invocar a função `session_start()` nos arquivos que precisam trabalhar com sessões. Esta função cria uma nova sessão (se não existir) ou resgata a sessão já criada (caso uma exista).

```
<?php
    session_start();
?>
```

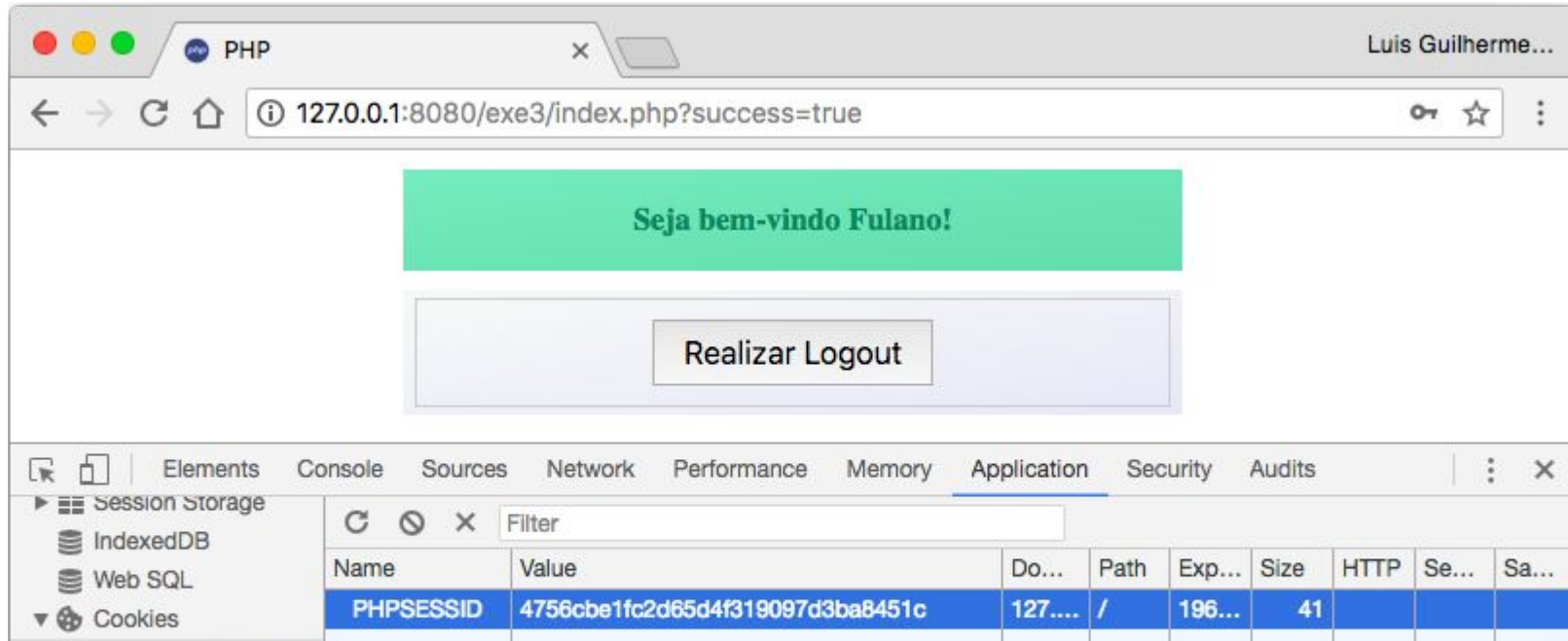
**Passo 2:** guardar dados na sessão através do acesso ao elemento `$_SESSION`, conforme código exposto a seguir.

```
session_start();
$_SESSION['usuario'] = $linha['nome'];
```

Neste caso, devemos usar `$_SESSION['nome']` para obter o valor, e `$_SESSION['nome'] = 'Fulano'` para definir um novo valor.

# Trabalhando com sessions (exe3/index.php)

O PHP guarda o ID da *session* em um *cookie* chamado **PHPSESSIONID**, que pode ser visualizado no *DevTools* do browser. Este ID é enviado a cada novo *Request*. Se o usuário apagá-lo, não será mais possível recuperar os dados guardados na *session* do servidor (obrigando o usuário a realizar um novo login).



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:8080/exe3/index.php?success=true`. The page content includes a green banner with the text "Seja bem-vindo Fulano!" and a button labeled "Realizar Logout". The DevTools Application tab is open, showing the Session Storage section. The table below lists the session data:

Name	Value	Do...	Path	Exp...	Size	HTTP	Se...	Sa...
PHPSESSID	4756cbe1fc2d65d4f319097d3ba8451c	127....	/	196...	41			

# Trabalhando com sessions (exe3/index.php)

Para apagar os dados da *session* deve-se utilizar a função *session\_destroy()*.

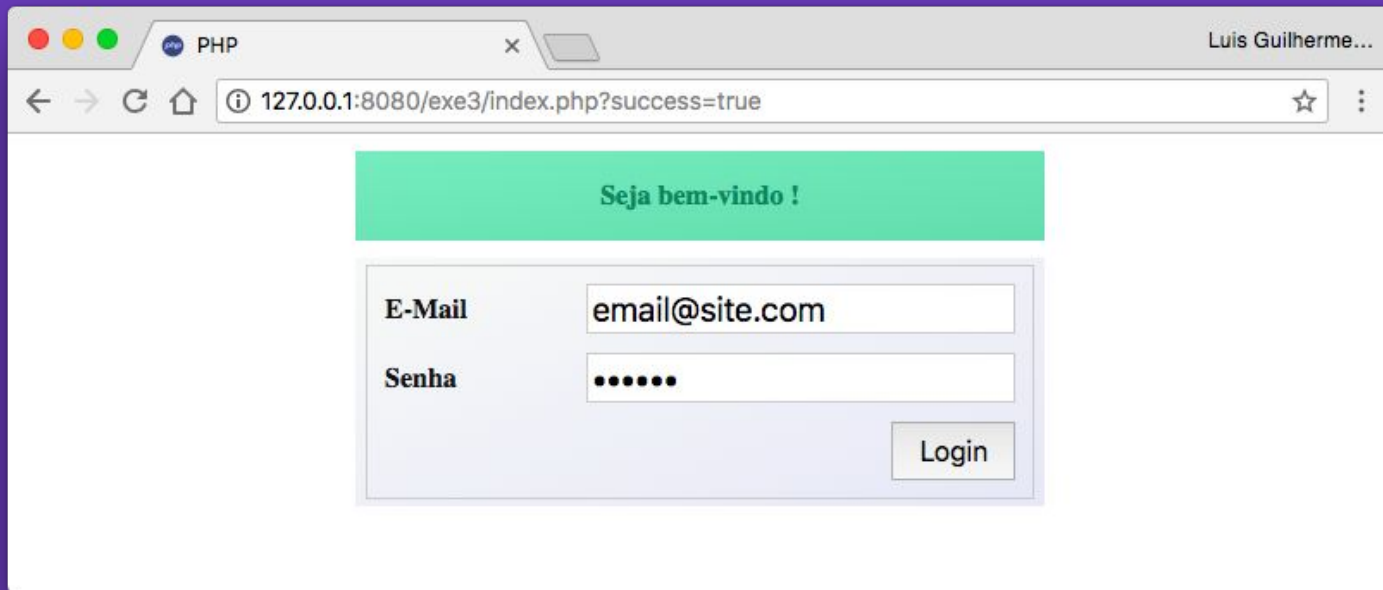
```
session_start();  
session_destroy();  
header('Location: index.php');  
die();
```

Note que é necessário executar também a função *session\_start()* antes da destruição da sessão para evitar *warnings* que o PHP exibe na tela caso o comando *session\_destroy()* seja executado sem existir uma sessão.



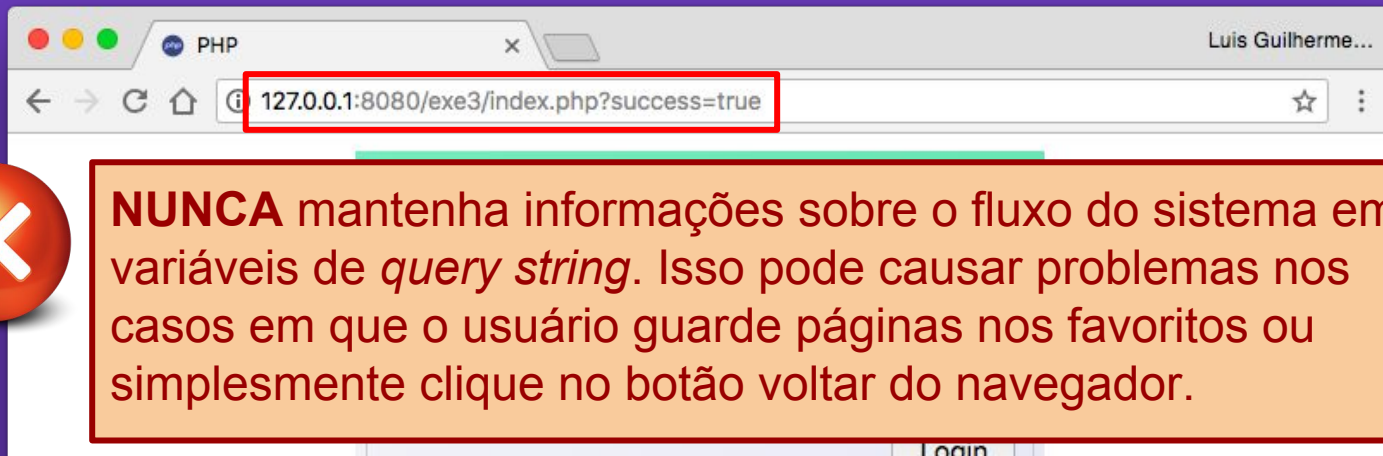
É uma boa prática criar scripts PHP de login e logout em arquivos específicos. Isso pode ser visualizado respectivamente nos arquivos *efetuar-login.php* e *efetuar-logout.php* do exercício 3.

Tente acrescentar a *query string* `?success=true` na página inicial do exercício 3...





Tente acrescentar a *query string* `?success=true` na página inicial do exercício 3...



**NUNCA** mantenha informações sobre o fluxo do sistema em variáveis de *query string*. Isso pode causar problemas nos casos em que o usuário guarde páginas nos favoritos ou simplesmente clique no botão voltar do navegador.

Então como devemos fazer?

# Mensagens nas sessions (exe4/index.php)

**Passo 1:** Sempre que houver necessidade de exibir uma mensagem ao usuário, crie uma variável na *session* para guardar a mensagem. Podemos inclusive guardar duas variáveis, uma para a mensagem de sucesso, outra para de erro.

```
session_start();  
$_SESSION['mensagem-sucesso'] = 'Logout realizado com sucesso!';
```

**Observação:** caso a sessão tenha sido destruída, crie uma nova.

**Passo 2:** na página de exibição, caso exista uma mensagem, após obtê-la da sessão, apague-a através da função *unset()*.

```
if ( isset($_SESSION['mensagem-erro']) ) {  
    $mensagemErro = $_SESSION['mensagem-erro'];  
    unset($_SESSION['mensagem-erro']);  
}
```

Dessa forma a mensagem nunca será exibida mais de uma vez.

**Guardando objetos na sessão**

# Objetos nas sessions (exe5/index.php)

Em sistemas maiores fica inviável (do ponto de vista da manutenibilidade) salvar variáveis separadas na sessão do usuário. Para este caso, podemos utilizar os métodos *serialize()* e *unserialize()* para, respectivamente, transformar o objeto em um formato "salvável", e converter este formato novamente em um objeto.

**Passo 1:** crie uma classe para o objeto que será salvo na sessão.

**Passo 2:** consulte os dados no banco de dados e instancie um novo objeto com esses dados.

**Passo 3:** guarde o objeto na sessão após serializá-lo com *serialize()*.

**Passo 4:** quando for obter o dado da sessão, utilize a função *unserialize()* para transformar o objeto novamente em uma instância da classe.

# Objetos nas sessions - PASSO 1 (exe5/index.php)

Crie a classe do objeto usuário

```
class Usuario {  
  
    private $codigo;  
    private $email;  
    private $nome;  
  
    public function __construct($codigo,$email,$nome) {  
        $this->codigo = $codigo;  
        $this->email = $email;  
        $this->nome = $nome;  
    }  
    public function getCodigo() {  
        return $this->codigo;  
    }  
    public function getEmail() {  
        return $this->email;  
    }  
    public function getNome() {  
        return $this->nome;  
    }  
}
```

# Objetos nas sessions - PASSO 2 (exe5/index.php)

Leia os dados no banco de dados e instancie o objeto.

```
$query = $conexao->prepare('select codigo, email, nome from usuarios where email = ? and senha = ?');  
$query->bind_param("ss",$email,hash('sha256', $senha));  
$query->execute();  
$resultado = $query->get_result();  
  
if ($resultado->num_rows == 1) {  
    $linha = $resultado->fetch_assoc();  
  
    $usuario = new Usuario(  
        $linha['codigo'],  
        $linha['email'],  
        $linha['nome']  
    );  
}
```



**Evite** trazer o campo de senha do usuário do banco de dados e **nunca** envie este campo para a camada cliente.

# Objetos nas sessions - PASSOS 3 e 4 (exe5/index.php)

*serialize()* o objeto e salve-o na sessão

```
$_SESSION['usuario'] = serialize($usuario);
```

*unserialize()* o objeto da sessão antes de utilizá-lo. Lembre-se de verificar primeiro se ele existe na sessão com a função *isset()*

```
$usuarioLogado = isset($_SESSION['usuario']);  
if ($usuarioLogado) {  
    $usuario = unserialize($_SESSION['usuario']);  
}
```

# Discussão em sala ([exe6/index.php](#))

Como faríamos para guardar na sessão os dados do perfil do usuário logado (além dos dados do próprio usuário)? Qual das opções abaixo vocês acham mais apropriada?

- Salvar um objeto perfil na sessão, assim como fizemos com o objeto usuário
- Salvar dentro do objeto usuário um atributo apontando para o objeto perfil. Dessa forma, o usuário da sessão já teria os dados do perfil.



# Objetos complexos nas sessions (exe6/index.php)

Para isso funcionar a SQL deve fazer join com a tabela de perfis

```
$query = $conexao->prepare(
    'select u.codigo, u.email, u.nome, u.codigoPerfil, p.descricao from usuarios u ' .
    'inner join perfis p on p.codigo = u.codigoPerfil ' .
    'where email = ? and senha = ?'
);
$query->bind_param("ss",$email,hash('sha256', $senha));
$query->execute();
$resultado = $query->get_result();
```

E primeiro deve-se criar o objeto perfil, para somente então criar o objeto usuário passando o objeto perfil por parâmetro.

Isso também requer um ajuste na classe *Usuario* e a criação da classe *Perfil*.

**Veja o código completo no exercício 6.**

```
$perfil = new Perfil(
    $linha['codigoPerfil'],
    $linha['descricao']
);

$usuario = new Usuario(
    $linha['codigo'],
    $linha['email'],
    $linha['nome'],
    $perfil
);

$_SESSION['usuario'] = serialize($usuario);
```

Obrigado!