

Autenticación de Usuarios

Manejo de Sesión

Problema

Tenemos nuestra app de tareas y el cliente nos solicita que ahora se pueda acceder solo con un **usuario y password**.



Pasos

¿Qué tenemos que agregar?

1. Mostrar un **formulario de login**.
2. Enviar el formulario y **verificar** que los datos del usuario son correctos.
3. Mantener al usuario **logueado** mientras navega el sitio.
4. Generar una acción para **desloguear** al usuario.



**KEEP
CALM
AND
WRITE
CODE**

Pasos - Form Login

1. Mostrar un **formulario de login**.

Página de Login

- MVC
 - **View** que muestre el formulario login
 - **Controller** que invoque a la vista
 - **Model?** Aún no es necesario



<https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/511179ca518b19c8be5c653bc2cc485b8b7b08b6>

Pasos - Form Login

2. Enviar el formulario y **verificar** que los datos del usuario son correctos.

- MVC
 - **Controller** que obtenga el usuario y verifique el pass.
 - **Model:** que obtenga el usuario que se quiere loguear.
 - **View:** que muestre un error si los datos son incorrectos.
- Tabla de ruteo
 - (POST) /verify

Tabla de Usuarios

- id
- email
- password



	#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Extra
<input type="checkbox"/>	1	id 🔑	int(11)			No	Ninguna	AUTO_INCREMENT
<input type="checkbox"/>	2	email	varchar(50)	latin1_swedish_ci		No	Ninguna	
<input type="checkbox"/>	3	password	varchar(255)	latin1_swedish_ci		No	Ninguna	

ENCRIPCIÓN DE CLAVES

**Las passwords NUNCA se almacenan
como texto plano**

¿Por qué?

Se debe usar algún algoritmo de hashing (md5, sha1, bcrypt, etc...)

El md5 de **123456** es:

e10adc3949ba59abbe56e057f20f883e

Los problemas con md5 y sha1 es que puede haber 2 passwords con el mismo hash.

password_hash()

PHP 5.5+ provee una función que nos ayuda a generar 'hashes' seguros.

Algoritmo por default bcrypt.

Modo de uso:

```
$hash = password_hash($password, PASSWORD_DEFAULT);
```

- El contenido de \$hash es lo que almaceno en la base de datos.
- PASSWORD_DEFAULT indica que algoritmo debe usar (el por defecto para passwords)

password_verify()

Modo de uso:

- Obtener el \$password del formulario que envía el usuario
- Obtener el \$hash de la base de datos

```
if (password_verify($password, $hash))  
    Credenciales válidas  
else  
    Credenciales invalidas
```



Resultado



BOLIVAR: <https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/9cb100b7686b3bdc588193fd749b426d728f36ec>

**¿Qué pasa si el usuario
conoce la URL de la página a
la que vamos después y la
tipea directamente?**



Diagrama de control de usuarios

Si un usuario intenta entrar a una URL a la que no tiene acceso lo redirigimos.
Vamos a llevarlo amablemente a la página de login.

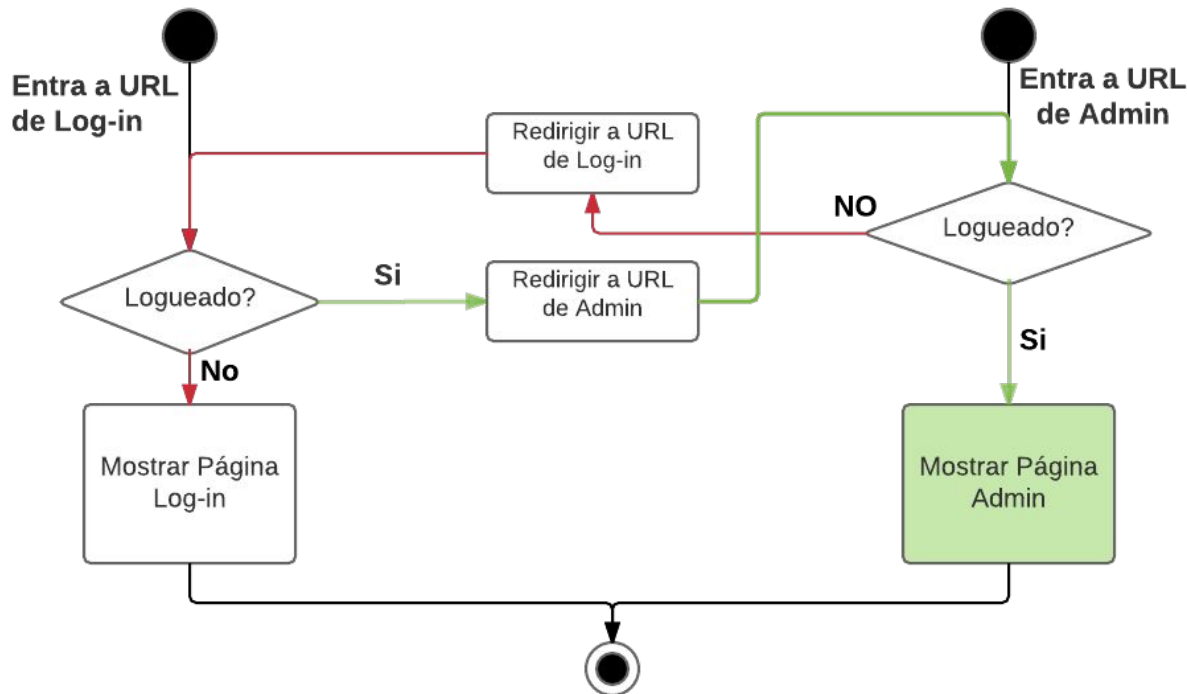
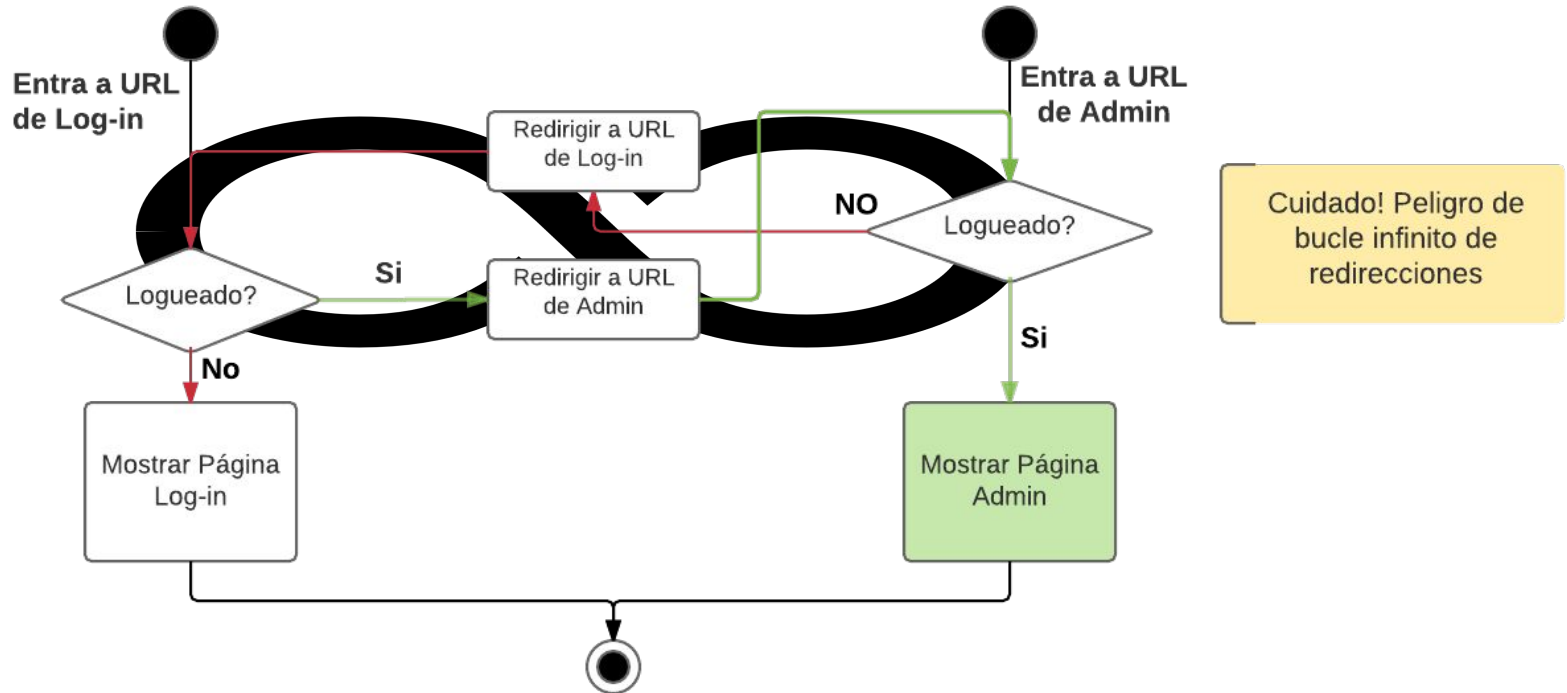


Diagrama de control de usuarios (2)

Si hago mal las comparaciones es fácil crear un bucle infinito de redirecciones que es difícil de ver y de arreglar



Sesión de Usuario

¿Cómo recordamos que el usuario ya **estaba logueado**?

SESIÓN DE USUARIO



Problemas de HTTP

HTTP es un **protocolo sin estado (state-less)**, es decir: no guarda ningún dato entre dos peticiones consecutivas.


- Cada request es totalmente independiente del anterior. Se crea una conexión y se cierra ni bien termina el pedido.
- Esto plantea la problemática, en caso de que los usuarios requieran interactuar con determinadas páginas web de forma ordenada y coherente.

¿Por qué creen que HTTP se diseñó state-less?

Opciones

El sitio web debería recordar que el usuario está logueado mediante algún mecanismo adicional.

Existen diferentes alternativas para **mantener a un usuario logueado** entre cada request.

- **Cookies**
 - **Local Storage**
 - **Session** → **SERVER SIDE**
- 
- The diagram uses a blue curly brace to group 'Cookies' and 'Local Storage' under the label 'CLIENT SIDE'. An arrow points from 'Session' to the label 'SERVER SIDE'.

Sesión de Usuarios - Client Side

COOKIES

Es una pequeña información enviada por un sitio web y almacenada por el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario.

<https://developer.mozilla.org/es/docs/Web/HTTP/Cookies>

LOCAL STORAGE

El almacenamiento local HTML5 se utiliza para almacenar pares de clave-valor en el lado del cliente. Estos pares de clave-valor se pueden recuperar en páginas HTML que provengan del mismo dominio. Los datos de almacenamiento local se almacenan en el disco y se conservan incluso después de reiniciar las aplicaciones.

<https://developer.mozilla.org/es/docs/Web/API/Window/localStorage>

HTML



Cookies - Solución

Guardar un estado de login en el cliente:

1. Cuando el usuario se loguea exitosamente, almaceno “algo” en el cliente (cookie o local storage).
*Por ejemplo, el nombre de usuario o un **flag** que ya está logueado.*
2. En cada http request le envié esta información al server.
3. El server usa esa información, la lee y determina que ya está logueado.

Sesión de Usuarios - Client Side

Cookies y Local Storage

- Viven del lado del cliente
- Cualquiera puede verlos.
- Son editables por los usuarios.

¿Ven algún problema con esto?

Si se conoce alguien con acceso, se puede estudiar la cookie de un usuario logueado para entender que tiene que poner en su cookie.

PHP Sesión - Server Side

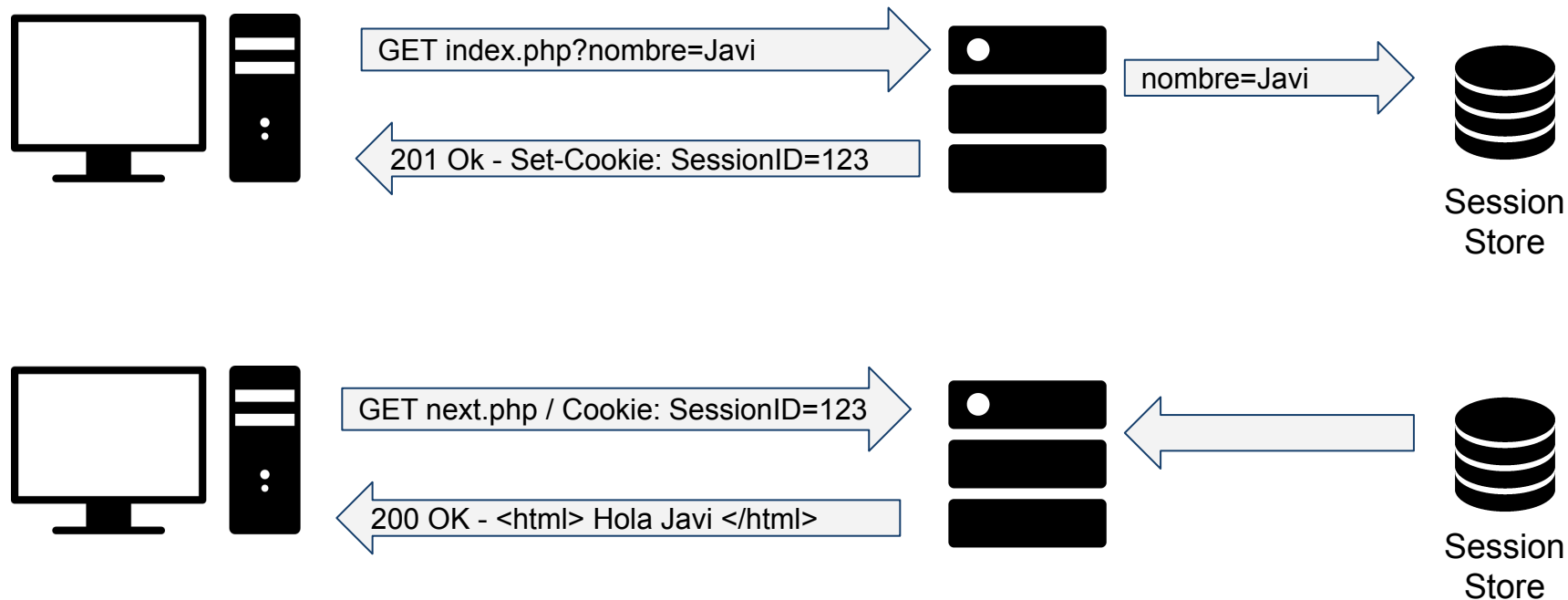
PHP SESSION

Utilizado para guardar información (estados) a través de los *requests* que un usuario hace durante su visita a un sitio web o aplicación.

La información guardada en una **sesión** puede llamarse en cualquier momento mientras la sesión esté abierta.

- La sesión **vive del lado del servidor**
- Es una pequeña porción de información que se guarda en el server
- Dura **mientras el usuario está conectado** al server
- Más seguro y confiable

Cómo funciona?



Manejo de Sesión en PHP

Con la sesión recuerdo el usuario para poder saber si estaba logueado.

```
<?php
```

```
session_start();
```

```
// sesión iniciada
```

- Crea una sesión en el servidor, si ya existe trae la existente.
- Debe **llamarse siempre** antes de acceder/almacenar algún dato.

Manejo de Sesión en PHP

`$_SESSION`

A través del arreglo asociativo `$_SESSION` se puede acceder en cualquier momento a los datos guardados en la sesión para el usuario conectado.

- Guardar una variable en la sesión

```
$_SESSION["nombre"] = "Web";
```

- Consultar si existe la sesión

```
isset($_SESSION["nombre"])
```

- Borrar un valor en la sesión

```
unset($_SESSION["nombre"])
```

Ejemplo PHP Session

[TBC]

Logueo al Usuario

Usamos la **SESSION** para mantener al usuario logueado:

```
public function verifyUser() {
    $username = $_POST['username'];
    $password = $_POST['password'];

    $user = $this->model->getByUsername($username);

    // encontró un user con el username que mandó, y tiene la misma contraseña
    if (!empty($user) && password_verify($password, $user->password)) {

        // INICIO LA SESSION Y LOGUEO AL USUARIO
        session_start();
        $_SESSION['ID_USER'] = $user->id;
        $_SESSION['USERNAME'] = $user->username;

        header('Location: ver');
    } else {
        $this->view->showLogin("Login incorrecto");
    }
}
```

Usuario denegado

Usamos la **SESSION** para determinar si el usuario ya estaba logueado:

```
/**
 * Muestra la lista de tareas.
 */
public function showTasks() {

    // barrera para usuario logueado
    $this->checkLoggedIn();

    // obtengo tareas del model
    $tareas = $this->model->getAll();

    // se las paso a la vista
    $this->view->showTasks($tareas);
}
```

```
private function checkLoggedIn() {
    session_start();
    if (!isset($_SESSION['ID_USER'])) {
        header('Location: ' . LOGIN);
        die();
    }
}
```

**QUE LA SESSION ESTÉ INICIADA NO
NECESARIAMENTE INDICA QUE EL USUARIO
ESTÁ LOGUEADO**

Redireccionar en PHP

- Para poder redireccionar desde PHP usamos la función `header()`.
- Tiene que ser usada **antes que se escriba algo por pantalla.**
- Luego de una redirección se suele llamar a la función `die()` para forzar terminar la ejecución del script.

```
header("Location: index.php");  
die();
```

```
define('HOME', 'http://'.$_SERVER['SERVER_NAME'] . dirname($_SERVER['PHP_SELF']).'/');  
define('LOGIN', 'http://'.$_SERVER['SERVER_NAME'] . dirname($_SERVER['PHP_SELF']).'/login');  
define('LOGOUT', 'http://'.$_SERVER['SERVER_NAME'] . dirname($_SERVER['PHP_SELF']).'/logout');
```

Resultado



BOLIVAR: <https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/f0f1a4a6de8808d37ce7f1f72be61926d772e650>

Logout

Logout

- Al desloguearse lo único que hay que hacer es borrar la información de la sesión.

session_destroy()

- Elimina la sesión borrando todas las variables almacenadas.
- Para eliminar la sesión siempre hay que iniciarla antes.

```
public function logout() {  
    session_start();  
    session_destroy();  
    header('Location: ' . LOGIN);  
}
```




**¿Cómo harían para que el
usuario se desloguee
automáticamente después de
media hora?**

Timeout

Debemos establecer un **tiempo de timeout** que destruya la sesión del usuario pasado cierto **tiempo de inactividad**.

Donde establecer el tiempo de timeout?

- COOKIES / LOCAL STORAGE:

Es manipulable por el usuario, podría modificarlo.

- SESSION:

Vive del lado del servidor, es seguro y confiable.

Timeout

Existen dos alternativas para manejar el tiempo de timeout del lado del servidor:

- Opciones de configuración de PHP

- [`session.gc_maxlifetime`](#)
- [`session.cookie_lifetime`](#)

} No son confiables, se deben generar mecanismos de control adicionales para que no modifique otros componentes del servidor

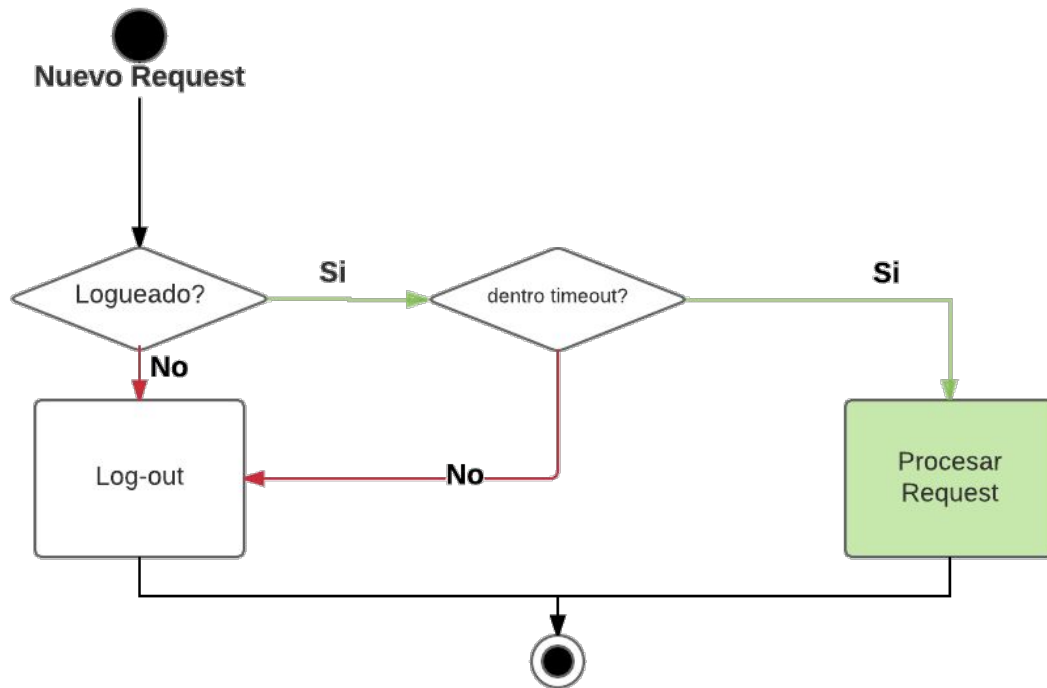
- Implementación manual del control de timeout

- Usar una marca de tiempo que determine el instante de la última actividad del usuario y verificar contra ella cada vez que el browser hace un request.



Timeout

Si el periodo de inactividad del usuario logueado fue mayor al timeout, lo redirigimos (otra vez amablemente) a la pantalla de login.



Timeout

Ejemplo:

```
// si el último instante de actividad fue hace más de 30 minutos
```

```
if ( isset($_SESSION['LAST_ACTIVITY']) &&  
    (time() - $_SESSION['LAST_ACTIVITY'] > 1800)) {  
    logout(); // destruye la sesión, y vuelve al login  
}
```

```
$_SESSION['LAST_ACTIVITY'] = time(); // actualiza el último instante de actividad
```

```
session_start();  
if(isset($_SESSION['USER'])){  
    if (time() - $_SESSION['LAST_ACTIVITY'] > 10) {  
        header('Location: '.LOGOUT);  
        die();  
    }  
    $_SESSION['LAST_ACTIVITY'] = time();  
}  
else {  
    header('Location: '.LOGIN);  
    die();  
}
```

Resultado



<https://gitlab.com/unicen/Web2/LiveCoding2018/Bolivar/EjemploTareas/commit/50642a7a60d6f648c30b2816284664863b6012af>

**¿Que pasa si se quiere verificar un
usuario logueado desde otro
Controller?**

**¿Que pasa si nos piden cambiar la
forma de manejar la session?**

Helpers

Clase auxiliar que proporciona funcionalidad similar y ayuda a resolver problemas comunes a lo largo de la aplicación.



Creamos **AuthHelper** para agrupar toda la funcionalidad de Autenticación.

AuthHelper (Resultado)



BOLIVAR: <https://gitlab.com/unicen/Web2/livecoding2019/bolivar/todo-list/commit/e8b30170631ba58784b15389981d1c846fa6fbc1>