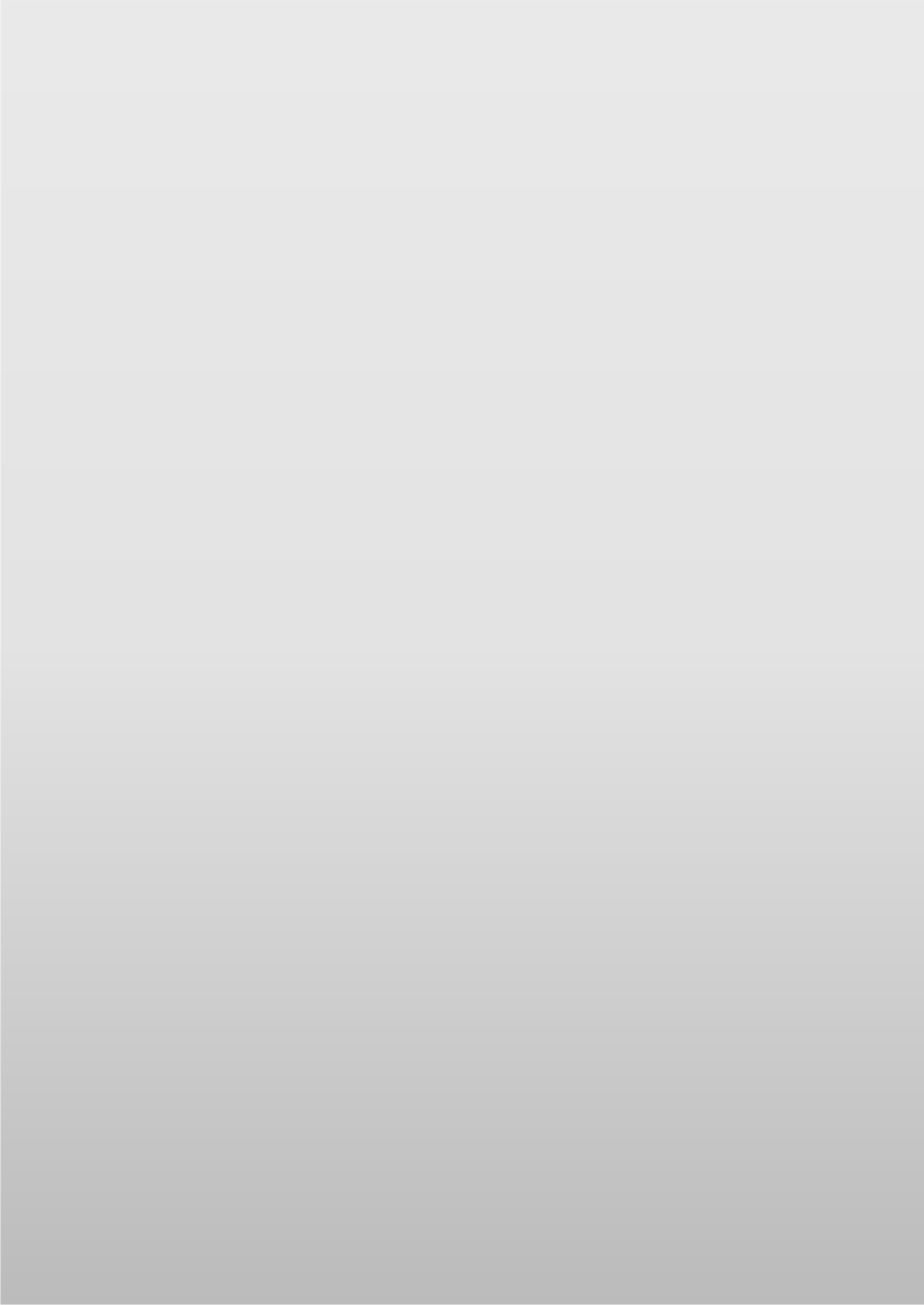
2025



UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

**PROGRAMACION WEB**

**GUIA DE USO Y APLICACIÓN**

**LocalStorage Y Cookies**

Imagen que contiene exterior, teléfono, parado, computadora

Descripción generada automáticamente

CUARTO SEMESTRE

NRC: 2260

**GUIA DE USO Y APLICACIÓN**

Local Storage y las cookies son dos mecanismos clave para el almacenamiento de datos en el navegador. Mientras que Local Storage permite guardar información de manera persistente en el dispositivo del usuario sin enviarla con cada solicitud HTTP, las cookies se utilizan principalmente para la gestión de sesiones y la comunicación con el servidor. Cada tecnología tiene sus ventajas y usos específicos, por lo que comprender sus diferencias y aplicaciones es fundamental para el desarrollo web.

# LOCAL STORAGE (Almacenamiento Local)

# Introducción

El local storage es una funcionalidad de los navegadores que permite a los desarrolladores guardar información en el navegador del usuario. Es parte de Web Storage API [API de Almacenamiento Web] junto a session storage. Funciona tomando información en formato de pares clave-valor y la conserva aun cuando el usuario actualiza la página o cierra la pestaña o el navegador.

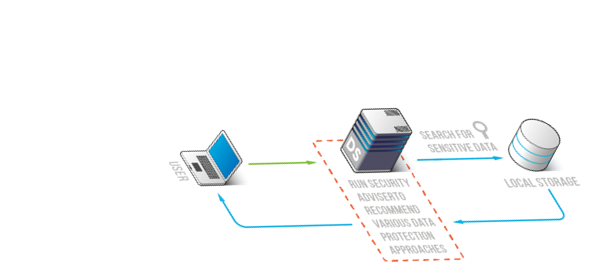
La [API de almacenamiento web](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API) es un conjunto de mecanismos que permite a los navegadores almacenar pares clave-valor. Está diseñada para ser mucho más intuitiva que el uso de cookies. El objeto de almacenamiento local ofrece distintos métodos que puede utilizar para interactuar con él. Con estos métodos, puede agregar, leer y eliminar datos del almacenamiento local.

# Local Storage en JavaScript

Es una propiedad que permite que los sitios y las aplicaciones de JavaScript guarden pares clave-valor en un navegador web sin fecha de vencimiento. Esto significa que los datos almacenados persisten incluso después de que el usuario cierra el navegador o reinicia la computadora.

*localStorage* es una *window* propiedad de objeto, lo que la convierte en un objeto global que puede interactuar con la ventana del navegador y manipularla. También se puede utilizar en combinación con otras window propiedades y métodos.

*localStorage* también permite a sus clientes acceder a datos específicos rápidamente sin la sobrecarga de una base de datos. Existen otras ventajas y desventajas que se deben comprender al trabajar con *localStorage*.



***Imagen 1*** *Ciclo de vida del TDA*

# *Window.localStorage*, ¿Qué es?

El mecanismo *localStorage* está disponible a través de la propiedad *Window.localStorage*. que es parte de la interfaz en JavaScript y representa una ventana que contiene un documento DOM.

La interfaz *Window* presenta una amplia gama de funciones, constructores, objetos y espacios de nombres. *Window.localStorage* es una propiedad de solo lectura que devuelve una referencia al objeto de almacenamiento local utilizado para almacenar datos a los que solo puede acceder el origen que los creó.

.

# ¿Cuándo usar *localStorage?*

Se utiliza para almacenar y recuperar datos. Si bien puede almacenar pequeñas cantidades de datos con *localStorage*, no es adecuado para grandes conjuntos de datos.

*localStorage* es accesible para cualquier persona que use el dispositivo, por lo que no debe usarlo para almacenar información confidencial. Puede usarlo para almacenar preferencias del usuario, como idioma o tema. También puede usarlo para almacenar datos en caché si lo usa con frecuencia. *localStorage* puede almacenar datos de formularios que no se perderán si el usuario cierra el navegador. Si tiene una aplicación que requiere que inicie sesión, *localStorage* puede usarla para guardar los datos de su sesión.

# Métodos de LocalStorage

Como se había mencionado antes *localStorage* almacena datos. Y, si lo haces, significa que es posible que necesites recuperarlos más adelante. En esta sección, exploraremos exactamente cómo localStoragefunciona

# Almacenamiento de datos, localStorage.setItem()

El método *setItem()* permite almacenar valores en *localStorage*. Toma dos parámetros: una clave y un valor. Se puede hacer referencia a la clave más adelante para obtener el valor asociado a ella. Así es como debería verse.

*localStorage.setItem(*'name', 'Obaseki Nosa'*);*

En el código anterior, puedes ver que *name* es la clave y *Obaseki Nosa* es el valor. Como ya hemos señalado, *localStorage* solo se pueden almacenar cadenas. Para almacenar matrices u objetos en *localStorage*, tendrías que convertirlos en cadenas.

Para eso usamos JSON.stringify() pasando el método before a setItem(), así:

const userArray = ["Obaseki",25]

localStorage.setItem('user', JSON.stringify(userArray));

# EJEMPLO IMPLEMENTACION

# Recuperando datos, localStorage.getItem()

La implementación de un TDA puede variar dependiendo del lenguaje de programación y del contexto en el que se utilice. Sin embargo, hay algunas pautas generales que se pueden seguir:

# Eliminar datos, localStorage.removeItem()

La implementación de un TDA puede variar dependiendo del lenguaje de programación y del contexto en el que se utilice. Sin embargo, hay algunas pautas generales que se pueden seguir:

# Eliminar todos los elementos, localStorage.clear ()

La implementación de un TDA puede variar dependiendo del lenguaje de programación y del contexto en el que se utilice. Sin embargo, hay algunas pautas generales que se pueden seguir:

***xxxxxx.html***

**#ifndef FECHA\_H #define FECHA\_H**

**class Fecha {**

**private:**

**int dia, mes, anio; public:**

**Fecha(int d, int m, int a); Fecha incrementar(int numDias);**

**int distancia(Fecha fin); // cambiar la variable periodo de**

**tiempo.**

**int obtenerMes();**

**private:**

**int diasEnMes(int m, int a);**

**int diasDesde1900(int d, int m, int a);**

**};**

***xxxxx.js***

**#endif**

**#include "Fecha.h"**

**Fecha::Fecha(int d, int m, int a) { dia = d;**

**mes = m; anio = a;**

**}**

**Fecha Fecha::incrementar(int numDias) { dia += numDias;**

**if (dia > diasEnMes(mes, anio)) { dia -= diasEnMes(mes, anio); mes++;**

**if (mes > 12) { mes = 1; anio++;**

**}**

**}**

**return Fecha(dia, mes, anio);**

**}**

# COOKIES

# Introducción

Las cookies son un elemento omnipresente en nuestra navegación por internet. Cada vez que visitamos una página web, es común encontrar un aviso que nos informa sobre el uso de cookies y nos solicita consentimiento para su almacenamiento en nuestros dispositivos.

# ¿Qué es una Cookie?

las cookies son pequeños archivos de texto que se guardan en el dispositivo de un usuario cuando visita un sitio web. Estos archivos contienen datos que el sitio puede leer durante la visita del usuario o en visitas posteriores. Esto permite a los sitios web recordar acciones y preferencias del usuario, lo que resulta esencial para proporcionar una experiencia de navegación personalizada y eficiente.

Es decir, aunque leyendo ciertos medios parezca que las cookies son *El Mal* —así, con mayúsculas—, en realidad fueron creadas para facilitar la navegación al usuario; por ejemplo, permiten recordar en qué idioma queremos navegar en una web o si nos hemos identificado con nuestro usuario y contraseña, para no tener que repetir estas acciones una y otra vez. Otro tema es que el uso de cookies de terceros haya llegado, en algunos casos, a límites que rozan el espionaje del usuario.



# Cookies según su plazo de tiempo

# Cookies de sesión

# Cookies persistentes

# Cookies seguras

Los TDAs son una abstracción en programación que encapsula datos y operaciones en una estructura cohesiva.

# Cookies según su finalidad

# Cookies Tecnicas u Obligatorias

# Cookies de preferencia o personalización

# Cookies de analítica o estadística

# Cookies de publicidad o marketing

Las plantillas permiten escribir código genérico que puede funcionar con múltiples tipos de datos sin necesidad de escribir implementaciones específicas para cada tipo.

.

# Cookies según el propietario:

# Cookies propias

# Cookies de tercero

# fjsnjd