## Clase 1

Contenido

[Clase 1 1](#_Toc174085914)

[Conceptos básicos 1](#_Toc174085915)

[Protocolo HTTP y su uso en aplicaciones cliente-servidor livianas 4](#_Toc174085916)

[Introducción al Protocolo HTTP 4](#_Toc174085917)

[Métodos de Petición HTTP (HTTP Verbs) 4](#_Toc174085918)

[Uso en Desarrollo de Aplicaciones Cliente-Servidor Livianas 5](#_Toc174085919)

[XAMPP: Un Entorno de Desarrollo Web Todo en Uno 5](#_Toc174085920)

[Aplicaciones sencillas en PHP 6](#_Toc174085921)

[Index.php 6](#_Toc174085922)

[Tabla del dos 7](#_Toc174085923)

[Saludo con formulario 7](#_Toc174085924)

[Arreglos indexados y ordinales 8](#_Toc174085925)

[Subarreglos 9](#_Toc174085926)

[Bibliografia 11](#_Toc174085927)

## Conceptos básicos

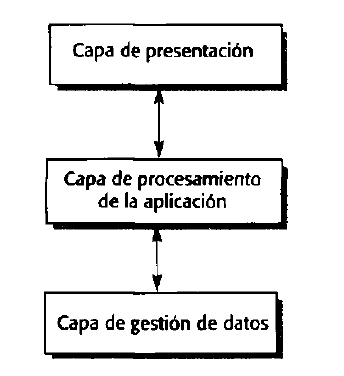
El modelo cliente-servidor es una arquitectura en la que un sistema se organiza en torno a un conjunto de servicios proporcionados por servidores y los clientes que acceden a estos servicios. Los componentes clave de este modelo incluyen:

1. **Servidores**: Estos proporcionan diversos servicios a otros subsistemas. Por ejemplo, puede haber servidores dedicados a la impresión, la gestión de archivos o la compilación de programas.
2. **Clientes**: Son los que utilizan los servicios ofrecidos por los servidores. Pueden existir múltiples instancias de un cliente ejecutándose al mismo tiempo, cada una accediendo a los servicios de los servidores.
3. **Red de comunicación**: Esta red facilita la interacción entre clientes y servidores. Aunque en teoría ambos pueden ejecutarse en la misma máquina, en la práctica suelen estar distribuidos en sistemas diferentes, formando un sistema distribuido.

Los clientes pueden conocer qué servidores están disponibles y qué servicios ofrecen, pero los servidores no necesitan saber quiénes son los clientes o cuántos están conectados. La interacción entre cliente y servidor se basa en un modelo de petición y respuesta, donde el cliente solicita un servicio y espera la respuesta del servidor.

Por ejemplo, en un sistema de biblioteca de películas y fotografías basado en la web, los servidores pueden manejar diferentes tipos de medios. Los vídeos, que requieren transmisión rápida, pueden ser manejados por un servidor específico que comprime y descomprime los formatos necesarios, mientras que las fotografías de alta resolución se almacenan en un servidor separado. El cliente, en este caso, sería una interfaz web que permite a los usuarios interactuar con estos servidores.

Una de las mayores ventajas de esta arquitectura es su naturaleza distribuida, que permite escalar fácilmente el sistema al añadir nuevos servidores o actualizar los existentes sin interrumpir el servicio a los clientes. Sin embargo, la integración de nuevos servidores puede requerir ajustes en los clientes y servidores existentes, especialmente si los datos se manejan de manera diferente en cada servidor. Aunque el uso de estándares como XML puede facilitar la conversión de datos entre diferentes formatos, esto podría impactar negativamente en el rendimiento debido a la ineficiencia de XML como formato de datos.

En una arquitectura cliente-servidor, una aplicación se estructura como un conjunto de servicios que son proporcionados por servidores, mientras que los clientes consumen estos servicios. Los clientes deben conocer qué servidores están disponibles, pero generalmente no interactúan ni son conscientes de otros clientes en el sistema. En este tipo de arquitectura, los clientes y los servidores son procesos separados, lo que significa que pueden ejecutarse en diferentes máquinas o en la misma máquina, pero en procesos distintos.

Una arquitectura cliente-servidor puede tomar diferentes formas, dependiendo de cómo se distribuyan las responsabilidades entre el cliente y el servidor. En una arquitectura de dos capas, que es la más simple, los servidores pueden manejar tanto la lógica de la aplicación como la gestión de datos, mientras que los clientes se encargan de la presentación al usuario. Este tipo de configuración puede encontrarse en dos variantes principales:

1. **Cliente Ligero**: En este modelo, el servidor asume casi toda la carga de trabajo, gestionando tanto la lógica de la aplicación como los datos. El cliente se encarga únicamente de la presentación, actuando como una interfaz gráfica para el usuario. Este enfoque es común en sistemas donde la infraestructura centralizada es heredada y la interfaz de usuario se ha migrado a dispositivos más modernos, como PCs o navegadores web.
2. **Cliente Rico**: Aquí, el cliente no solo presenta la información, sino que también procesa parte de la lógica de la aplicación. El servidor, en este caso, se concentra principalmente en la gestión de los datos. Un ejemplo típico es el sistema de cajeros automáticos (ATM), donde cada cajero procesa parte de la información antes de enviarla al servidor principal.

Cuando se desarrolla una arquitectura cliente-servidor, es crucial que el diseño refleje la estructura lógica de la aplicación. En sistemas distribuidos, estas arquitecturas se pueden organizar en varias capas, lo que facilita su distribución en diferentes máquinas, optimizando así el rendimiento y la escalabilidad del sistema. Por ejemplo, en una arquitectura de tres capas, las tareas de presentación, lógica de la aplicación y gestión de datos se separan, permitiendo que cada una se ejecute en un servidor diferente, lo que mejora la eficiencia del sistema.

Este tipo de arquitectura, particularmente en su variante de tres capas, es más escalable y manejable que la de dos capas. En contextos donde las aplicaciones deben interactuar con múltiples bases de datos o donde se necesita un alto rendimiento, esta separación de responsabilidades permite que cada capa se optimice y se gestione de forma independiente, lo que facilita la actualización y el mantenimiento del sistema.[1]

## Protocolo HTTP y su uso en aplicaciones cliente-servidor livianas

### Introducción al Protocolo HTTP

El **Protocolo de Transferencia de Hipertexto (HTTP)** es un protocolo de la capa de aplicación utilizado principalmente para la transmisión de documentos hipermedia, como páginas web en formato HTML. Es un protocolo esencial para la comunicación en la web, permitiendo que los navegadores (clientes) soliciten y reciban datos de los servidores. HTTP sigue un modelo de comunicación clásico cliente-servidor, donde un cliente establece una conexión con el servidor, realiza una petición y espera una respuesta.

Una característica importante de HTTP es que es un protocolo **sin estado**, lo que significa que el servidor no retiene información entre diferentes peticiones del cliente. Cada solicitud es tratada de forma independiente, lo cual simplifica la gestión del servidor, pero también puede requerir técnicas adicionales para mantener sesiones o estados en aplicaciones más complejas.

Aunque comúnmente se usa sobre una conexión TCP/IP, HTTP puede funcionar sobre cualquier capa de transporte que asegure la entrega de mensajes, como UDP, siempre que se garantice la confiabilidad [2].

### Métodos de Petición HTTP (HTTP Verbs)

HTTP define varios métodos de petición, conocidos como HTTP verbs, que determinan la acción a realizar sobre un recurso en el servidor. A continuación, se describen los métodos más comunes:

GET: Solicita una representación de un recurso específico. Este método es el más utilizado para recuperar datos y es considerado seguro, ya que no modifica el estado del servidor.

HEAD: Similar a GET, pero solo solicita los encabezados de la respuesta sin el cuerpo. Es útil para verificar si un recurso está disponible sin descargar su contenido.

POST: Se utiliza para enviar datos al servidor para crear o modificar un recurso. A menudo causa un cambio en el estado del servidor o efectos secundarios, como el envío de un formulario.

PUT: Reemplaza todas las representaciones actuales de un recurso con los datos enviados en la solicitud. Es idempotente, lo que significa que realizar la misma operación varias veces produce el mismo resultado.

DELETE: Elimina un recurso específico del servidor.

CONNECT: Establece un túnel hacia el servidor, normalmente utilizado para conexiones seguras, como HTTPS.

OPTIONS: Devuelve las opciones de comunicación disponibles para un recurso, lo que permite a los clientes entender qué métodos pueden ser utilizados.

TRACE: Realiza un test de bucle de retorno para diagnosticar el camino que sigue la solicitud hasta el servidor.

PATCH: Aplica cambios parciales a un recurso, lo cual es útil para actualizaciones incrementales en lugar de reemplazos completos.

### Uso en Desarrollo de Aplicaciones Cliente-Servidor Livianas

En el desarrollo de aplicaciones cliente-servidor livianas, HTTP es especialmente ventajoso debido a su simplicidad y su amplio soporte en la infraestructura de Internet. Las aplicaciones web ligeras suelen emplear el método GET para recuperar datos desde el servidor y POST para enviar datos del cliente al servidor, como en el caso de formularios. El enfoque sin estado de HTTP permite a los desarrolladores construir aplicaciones que pueden escalar fácilmente, ya que cada solicitud es independiente, facilitando la distribución de carga entre múltiples servidores.

Además, la combinación de HTTP con tecnologías como AJAX permite a las aplicaciones cliente-servidor livianas interactuar de manera asíncrona, mejorando la experiencia del usuario al permitir actualizaciones en tiempo real sin recargar la página completa.

## XAMPP: Un Entorno de Desarrollo Web Todo en Uno

XAMPP es un paquete de software libre que facilita la instalación y configuración de un entorno de desarrollo web completo en tu computadora. Su nombre es un acrónimo que representa las principales herramientas que incluye: X (cualquier sistema operativo), Apache (servidor web), MariaDB/MySQL (sistema de gestión de bases de datos), PHP y Perl (lenguajes de script).

Este paquete está diseñado para simplificar el proceso de configuración de un servidor local, permitiendo a desarrolladores y diseñadores web probar y desarrollar sus proyectos sin necesidad de estar conectados a Internet. XAMPP es fácil de instalar y usar, ya que agrupa todos los componentes necesarios en un solo paquete, eliminando la necesidad de descargar e instalar cada herramienta por separado.

En nuestras clases de laboratorio, utilizaremos XAMPP para crear un entorno de desarrollo local donde podremos probar aplicaciones web, trabajar con bases de datos y desarrollar scripts en PHP. Su sencilla interfaz y su capacidad para ejecutar páginas dinámicas lo hacen ideal para aprender y practicar en un entorno controlado[3].



## Aplicaciones sencillas en PHP

### Index.php

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Document</title>

</head>

<body>

<input type="email" name="" id="">

<?php

$nombre="Plácido Maidana";

echo "Hola    ", $Nombre , 2 . '<input type="email" name="" id="">';

?>

<h1>Hola mundo</h1>

</body>

</html>

### Tabla del dos

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <title>Tabla de Multiplicar del 2</title>

</head>

<body>

    <h1>Tabla de Multiplicar del 2</h1>

    <ul>

        <?php

        for ($i = 1; $i <= 10; $i++) {

            echo "<li>2 x $i = " . (2 \* $i) . "</li>";

        }

        ?>

    </ul>

</body>

</html>

### Saludo con formulario

<!DOCTYPE html>

<html lang="es">

<head>

    <meta charset="UTF-8">

    <title>Saludo Personalizado</title>

</head>

<body>

    <h1>Formulario de Saludo</h1>

    <?php

    if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

        $nombre = htmlspecialchars($\_POST["nombre"]);

        echo "<h2>Hola, $nombre!</h2>";

    } else {

    ?>

    <form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]); ?>">

        <label for="nombre">Ingresa tu nombre:</label>

        <input type="text" id="nombre" name="nombre" required>

        <input type="submit" value="Enviar">

    </form>

    <?php

    }

    ?>

</body>

</html>

### Arreglos indexados y ordinales

Arreglos 1

<?php

// Arreglo ordinal de días de la semana en los que vas al gimnasio

$diasGimnasio = ["Lunes", "Miércoles", "Viernes"];

// Imprimir los días en los que vas al gimnasio

echo "Días en los que voy al gimnasio:<br>";

foreach ($diasGimnasio as $dia) {

    echo $dia . "<br>";

}

// Arreglo bidimensional para almacenar números generados aleatoriamente

$numeros = [];

// Función para llenar el arreglo bidimensional con números aleatorios

function llenarArregloBidimensional(&$arreglo, $filas, $columnas) {

    for ($i = 0; $i < $filas; $i++) {

        for ($j = 0; $j < $columnas; $j++) {

            // Generar número aleatorio entre 1 y 100

            $arreglo[$i][$j] = rand(1, 100);

        }

    }

}

// Llenar el arreglo bidimensional de 3x3

llenarArregloBidimensional($numeros, 3, 3);

// Imprimir el arreglo bidimensional

echo "<br>Números generados aleatoriamente:<br>";

for ($i = 0; $i < count($numeros); $i++) {

    for ($j = 0; $j < count($numeros[$i]); $j++) {

        echo $numeros[$i][$j] . " ";

    }

    echo "<br>";

}

/\*

  Diferencias entre arreglos asociativos y ordinales:

  - Arreglos ordinales (o indexados): son aquellos en los que las claves son números enteros secuenciales (0, 1, 2, etc.). En el ejemplo, $diasGimnasio es un arreglo ordinal.

  - Arreglos asociativos: son aquellos en los que las claves son cadenas de texto. Se utilizan cuando queremos asociar valores con claves específicas, no numéricas.

  Ejemplo de un arreglo asociativo:

\*/

$asociativo = [

    "Lunes" => "Ir al gimnasio",

    "Martes" => "Descanso",

    "Miércoles" => "Ir al gimnasio",

];

// Imprimir los valores del arreglo asociativo

echo "<br>Actividades de la semana:<br>";

foreach ($asociativo as $dia => $actividad) {

    echo $dia . ": " . $actividad . "<br>";

}

?>

### Subarreglos

Crea un arreglo asociativo donde cada día de la semana tiene un subarreglo de horas del día. En este subarreglo, se especifica el tiempo que estarás en el trabajo y el tiempo que estarás en el gimnasio.

<?php

// Arreglo asociativo de días de la semana con subarreglos de horas del día

$horarios = [

    "Lunes" => [

        "Trabajo" => ["08:00 - 12:00", "13:00 - 17:00"],

        "Gimnasio" => ["18:00 - 19:00"]

    ],

    "Martes" => [

        "Trabajo" => ["08:00 - 12:00", "13:00 - 17:00"],

        "Gimnasio" => ["19:00 - 20:00"]

    ],

    "Miércoles" => [

        "Trabajo" => ["08:00 - 12:00", "13:00 - 17:00"],

        "Gimnasio" => ["18:00 - 19:00"]

    ],

    "Jueves" => [

        "Trabajo" => ["08:00 - 12:00", "13:00 - 17:00"],

        "Gimnasio" => ["19:00 - 20:00"]

    ],

    "Viernes" => [

        "Trabajo" => ["08:00 - 12:00", "13:00 - 17:00"],

        "Gimnasio" => ["18:00 - 19:00"]

    ],

    "Sábado" => [

        "Trabajo" => ["09:00 - 13:00"],

        "Gimnasio" => ["17:00 - 18:00"]

    ],

    "Domingo" => [

        "Trabajo" => ["Libre"],

        "Gimnasio" => ["10:00 - 11:00"]

    ]

];

// Mostrar el horario de cada día

foreach ($horarios as $dia => $actividades) {

    echo "<strong>$dia:</strong><br>";

    foreach ($actividades as $actividad => $horas) {

        echo "$actividad: ";

        echo implode(", ", $horas); // Unir las horas en una sola cadena separadas por comas

        echo "<br>";

    }

    echo "<br>";

}

/\*

  En este código:

  - Cada día de la semana es una clave en el arreglo asociativo $horarios.

  - El valor asociado a cada día es otro arreglo asociativo, donde las claves son "Trabajo" y "Gimnasio", y los valores son arreglos que contienen las horas correspondientes.

  - Este diseño permite almacenar múltiples rangos de horas para cada actividad dentro de un mismo día.

\*/

?>

## Bibliografia

[1] Sommerville, *Ingeniería del software Ingeniería del software seguridad*. 2005.

[2] MDN Web Docs, “Métodos de petición HTTP,” *https://developer.mozilla.org/*, 2023, [Online]. Available: https://developer.mozilla.org/es/docs/Web/HTTP/Methods

[3] J. Ibero, “Arquitectura Cliente/Servidor: modelo de 3 capas,” *IberAsync.es*, 2024, [Online]. Available: https://iberasync.es/arquitectura-cliente-servidor-modelo-de-3-capas/