

Material Programación Full Stack

Índice

Arquitectura de software Cliente-Servidor	3
Stack tecnológico	4
Frontend	4
Stack tecnológico de frontend	4
Librerías	5
Bootstrap	5
Guía de Bootstrap	5
JavaScript	5
Guía de JavaScript	5
JQuery	6
Backend	6
Stack tecnológico de backend (LAMP)	6
Desarrollador Full Stack	7
¿Qué es una API REST?	7
Protocolo HTTP	7
¿Qué es un método HTTP? (Verbo)	7
GET	7
POST	7
PUT	8
DELETE	8
OWASP - Pruebas y verificación	8
Oauth 2.0	11
Hash	11
MD5	11
Guía de GIT y flujo de trabajo centralizado	12
Containerización	12
Contenedores y máquinas virtuales	12
Docker	13
Docker Compose	13
docker-compose.yaml	13
Plan de testing	14
Ejemplo	14
Pruebas de caja negra	14
Pruebas funcionales	14
Pruebas unitarias	14
Pruebas de regresión	14
Pruebas no funcionales	14
Pruebas de caja blanca	14

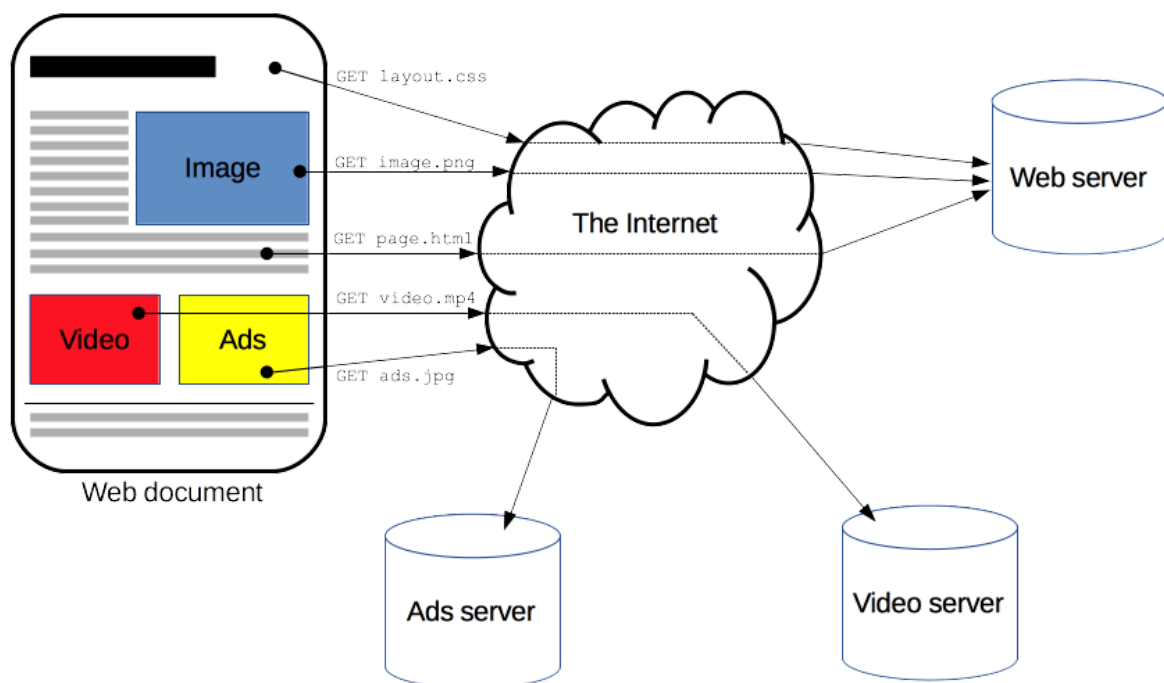
Lenguaje de programación PHP	15
Hola Mundo en PHP	15
Elementos y funciones básicas de PHP	15
Comentarios	15
Variables	15
Concatenar	16
Operaciones aritméticas	16
Salto de línea	17
Número flotante	17
Redondear un float	17
Número aleatorio	17
Tipo de dato booleano	18
Arreglo en PHP	18
Sesiones en PHP	19
Sitio web Multilenguaje	19
Servidor web interno de PHP	20
Material teórico, ejemplos y ejercicios de Bases de datos	20
Aplicaciones de ejemplo	21
1. Subir archivo a servidor con PHP. 🔗 Enlace	21
2. Login y logout de usuario con PHP (proyecto de ejemplo) 🔗 Enlace	21
3. Conexión y consulta a base de datos MySQL	21
4. API Restful y Solicitudes desde JavaScript 🔗 Enlace	22
5. Sitio web Multilenguaje	22
a. Desde archivos PHP 🔗 Enlace	22
b. Desde API y con Javascript 🔗 Enlace	22
6. Seguridad: Login con API (Token/KEY OAuth 2.0) 🔗 Enlace	22
7. Testing: Pruebas automatizadas de funcionalidades de API 🔗 Enlace	23
8. API Restful y Solicitudes desde videojuego en JavaScript 🔗 Enlace	23
9. Seguridad: Ataque XSS. 🔗 Enlace	24
10. Seguridad: Inyección SQL. 🔗 Enlace	24
X. Seguridad: Sesiones y cookies.	24
X. (GET) Consumir API restful	24
X. Crear API restful	24
X. (POST) Consumir API restful desde formulario html	25
X. (DELETE) Consumir API restful desde formulario html	26
X. (PUT) Consumir API restful desde formulario html	27
11. Login de usuario a través de API	28
12. (OAuth 2.0) Login de usuario a través de API con token	32
13. (Docker) Despliegue de aplicación con contenedores	36
14. Lectura y escritura de archivo de texto	42
15. Verificación de existencia y creación de directorio en el servidor	42

Arquitectura de software Cliente-Servidor

Es un modelo de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.

Se comunican intercambiando mensajes, los que envía el cliente se llaman *peticiones*, y los enviados por el servidor se llaman *respuestas*. El cliente (normalmente en un navegador Web) es el que inicia una comunicación (petición), no es común que el servidor la comience ya que este solo se limita a responder las peticiones. Un servidor no tiene que ser necesariamente un único equipo físico.

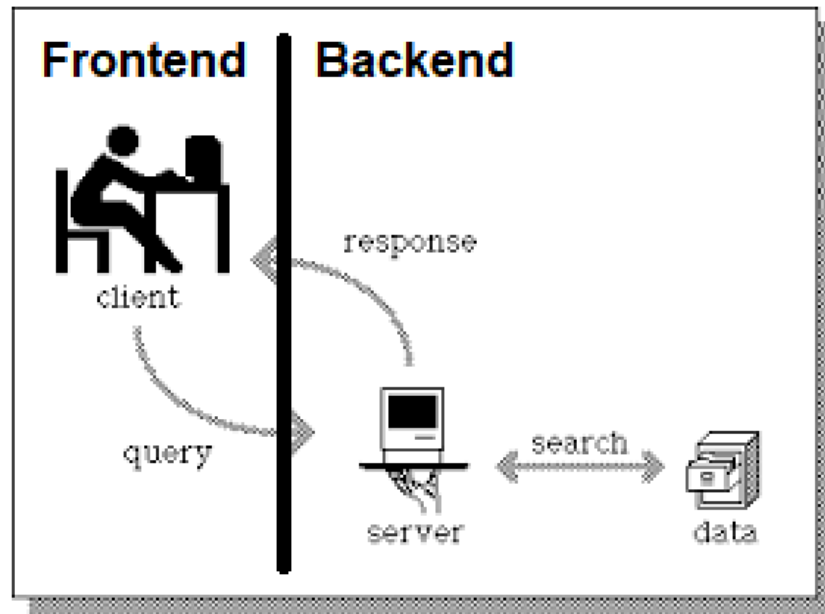
Para poder mostrar una página Web, el navegador envía una petición de documento HTML al servidor. Al recibirlo, procesa el documento y envía más peticiones para solicitar scripts, hojas de estilo (CSS), y otros datos que necesite (normalmente imágenes y/o vídeos). El navegador une todos estos documentos y datos, y compone el resultado final: la página Web.



Stack tecnológico

Es la colección de herramientas, plataformas, aplicaciones y piezas de software que una empresa utiliza para crear y mantener sus productos.

Un sitio web puede clasificarse de diferentes formas. Para cuestiones de desarrollo web principalmente se divide en dos partes.



Frontend

Es la parte que interactúa con el usuario, tanto en imagen como en función. Por ello está íntimamente relacionada con la experiencia del usuario (UX) y la interfaz de usuario (IU).

Stack tecnológico de frontend

Los desarrolladores frontend se enfocan en la codificación y programación del sitio web utilizando lenguajes como HTML, CSS, JavaScript y otros. Su objetivo es que el sitio responda correctamente a las interacciones que realiza el usuario en él.



Librerías

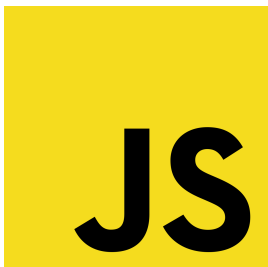
Las librerías son códigos reutilizables que proporcionan funcionalidades estándar para que los desarrolladores no tengan que escribirlas desde cero. Una librería consta de varias funciones, objetos y métodos que pueden interactuar con componentes de la interfaz de usuario o realizar funciones matemáticas.



Bootstrap

Es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como extensiones de JavaScript adicionales. Facilita la creación de diseños responsivos, adaptables a diferentes dispositivos.

[Guía de Bootstrap](#)



JavaScript

Es un lenguaje de programación ampliamente utilizado para crear interactividad y dinamismo en páginas web. Se ejecuta directamente en el navegador del usuario, permitiendo actualizar contenido, animaciones y funcionalidades sin necesidad de recargar la página.

[Guía de JavaScript](#)



jQuery

Es una librería de JavaScript rápida e incluida en un solo archivo. Proporciona muchas funciones integradas mediante las cuales puedes realizar diversas tareas de manera fácil y rápida, como selección o manipulación DOM. Permite agregar interactividad y efectos visuales en un sitio web.

Backend

Se refiere a la parte que está en contacto directo con el servidor; es donde se aplica el código de programación para crear la lógica de negocio. Permanece en un segundo plano a cargo de la accesibilidad, actualización, bases de datos y cambios del sitio.

Stack tecnológico de backend (LAMP)

Es el acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas:

- **Linux**, el sistema operativo.
- **Apache**, el servidor web.
- **MySQL/MariaDB**, el gestor de bases de datos.
- **PHP**, el lenguaje de programación.

La combinación de estas tecnologías es usada principalmente para definir la infraestructura de un servidor web, a pesar de que el origen de estos programas de código abierto no fue específicamente diseñado para trabajar entre sí. Cuando son combinados, representan un conjunto de soluciones que proporcionan los servidores de aplicaciones. Es fácil de aprender e implementar. Existen una cantidad enorme de recursos, tutoriales, cursos, libros sobre las tecnologías y es casi un estándar en el desarrollo web.



Desarrollador Full Stack

Un Full Stack Developer es aquel profesional que domina las tecnologías de las dos partes de un desarrollo web, es decir, tanto los aspectos del frontend, la parte del código con la que el usuario puede interactuar en un navegador, como los del backend, la gestión interna de la página y de la comunicación entre el servidor y la base de datos. Además de los lenguajes de programación, tiene que saber controlar los diferentes sistemas operativos, bases de datos y servidores.

¿Qué es una API REST?

Es una interfaz que los sistemas utilizan para intercambiar información de manera segura a través de Internet. Las aplicaciones se comunican con las API a través de métodos HTTP (como GET o POST) para acceder y manipular recursos.

REST (Transferencia de Estado Representacional) es un estilo arquitectónico para diseñar sistemas de software distribuidos, como servicios web, que prioriza la simplicidad, la escalabilidad y la facilidad de uso. En esencia, REST define un conjunto de principios para construir API que interactúan a través de HTTP, utilizando recursos identificados por URLs y manipulados mediante métodos como GET, POST, PUT y DELETE.

Protocolo HTTP

Es un protocolo de capa de aplicación diseñado para transferir información, se ejecuta sobre otras capas del conjunto de protocolos de la red. Un flujo típico sobre HTTP implica que un cliente realiza una solicitud a un servidor, que en consecuencia envía un mensaje de respuesta.

Una solicitud HTTP es la forma en que las plataformas de comunicación de Internet, como los navegadores web, solicitan la información necesaria para cargar el contenido de un sitio web. Cada solicitud HTTP realizada a través de Internet lleva consigo una serie de datos codificados que contienen diferentes tipos de información.

¿Qué es un método HTTP? (Verbo)

Un método HTTP, a veces denominado verbo HTTP, indica la acción que la solicitud HTTP espera del servidor consultado. Por ejemplo, dos de los métodos HTTP más comunes son "GET" y "POST". Los verbos Http involucrados en un sistema REST son GET, POST, PUT, y DELETE.

GET

Es el que usamos para consultar un recurso. Una de las principales características de una petición GET es que no debe causar efectos secundarios en el servidor, no deben producir nuevos registros, ni modificar los ya existentes.

POST

Las peticiones con POST son sólo para crear recursos nuevos. Cada llamada con POST debería producir un nuevo recurso.

Normalmente, la acción POST se dirige a una recurso que representa una colección, para indicar que el nuevo recurso debe agregarse a dicha colección, por ejemplo POST /cursos para agregar un nuevo recurso a la colección *cursos*.

PUT

El verbo PUT se usa para modificar un recurso existente o sustituirlo por completo.

DELETE

Este verbo se utiliza para eliminar registros.

OWASP - Pruebas y verificación

Es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro. Este proyecto es apoyado y gestionado por la fundación OWASP (Open Web Application Security Project), que es un organismo sin ánimo de lucro.

¿Por qué es necesario probar la seguridad?

El software inseguro es quizás el reto técnico más importante de nuestros tiempos. La seguridad es ahora el factor clave limitante sobre lo que podemos crear con la tecnología de la información. OWASP intenta hacer del mundo un lugar en el que el software inseguro es la anomalía, no la norma, y la Guía de Pruebas es una pieza importante del rompecabezas.

Existe un número infinito de modos en que un atacante podría ser capaz de colgar una aplicación, y es simplemente imposible comprobarlas todos. Sin embargo, la comprobación de seguridad ha demostrado ser un elemento clave para cualquier organización que necesita confiar en el software que produce o usa.

¿A quienes está dirigida la guía OWASP?

Desarrolladores de Software: Necesitan usar la guía para asegurarse que el código que se entrega no es vulnerable a ataques. Los testers o grupos de seguridad nunca entenderán su aplicación tan bien como usted, y por lo tanto nunca serán capaces de probar su aplicación tan efectivamente como usted puede. ¡La responsabilidad de la seguridad de su código es suyo!.

Testers de Software

Deberían usar esta guía para mejorar sus habilidades para probar. Mientras las pruebas de seguridad han sido artes oscuras por un largo tiempo, OWASP está trabajando duro para hacer este conocimiento gratuito y abierto para todos. Muchas de las pruebas descritas en esta guía no son complicados y no requieren habilidades especiales o herramientas. Puede ayudar a su compañía y mejorar su carrera al aprender sobre seguridad.

Especialistas de Seguridad

Tiene una responsabilidad especial para asegurar que las aplicaciones no se publiquen con vulnerabilidades. Puede usar esta guía para ayudar a asegurar un nivel de cobertura y rigor. Su trabajo es verificar la seguridad de la aplicación completa.

ASVS (Application Security Verification Standard)

El estándar de verificación de seguridad en aplicaciones cuenta con distintos niveles de verificación.

- **ASVS Nivel 1:** es para bajos niveles de garantía, y es completamente comprobable con pentesting.
- **ASVS Nivel 2:** es para aplicaciones que contienen datos confidenciales, que requiere protección y es el nivel recomendado para la mayoría de las aplicaciones.
- **ASVS Nivel 3:** es para las aplicaciones más críticas - aplicaciones que realizan transacciones de alto valor, contienen datos médicos sensibles, o cualquier aplicación que requiere el más alto nivel de confianza.

Se recomienda utilizar el estándar de verificación de seguridad en aplicaciones como una guía para crear una lista de comprobación de codificación segura específica para su aplicación, plataforma u organización. Adaptar el ASVS a sus casos de uso aumentará el enfoque en los requisitos de seguridad que son más importantes para sus proyectos y entornos.

¿Cómo hacer referencia a los requisitos de ASVS?


Cada requisito tiene un identificador en el formato <chapter>.<section>.<requirement> donde cada elemento es un número, por ejemplo: 1.11.3.

- El elemento <chapter> corresponde al capítulo del que proviene el requisito, por ejemplo: todos los requisitos de 1.## son del capítulo de "Arquitectura".
- El elemento <section> corresponde a la sección dentro de ese capítulo donde aparece el requisito, por ejemplo: todos los requisitos de 1.11.# están en la sección "Arquitectura de la Lógica del Negocio", del capítulo de "Arquitectura".
- El elemento <requirement> identifica el requisito específico dentro del capítulo y la sección, por ejemplo: 1.11.3 que a partir de la versión 4.0.3 del presente estándar es: el 3er requisito en la sección "Arquitectura de la Lógica del Negocio" del capítulo "Arquitectura".

Es también común encontrar los identificadores precedidos por la versión del estándar que se utiliza. Por ejemplo para la versión 4.0.3 los identificadores pueden registrarse de la siguiente manera: v4.0.3-1.11.3.

Requisitos de ASVS

En el siguiente enlace encontrarás la lista de requisitos del estándar de verificación de seguridad de aplicaciones (ASVS) en distintas categorías, utiliza los requisitos que apliquen a tu proyecto.

 [OWASP Application Security Verification Standard 4.0.3-es.pdf](#)

V1.1 Ciclo de Vida de Desarrollo de Software Seguro

#	Descripción	L1	L2	L3	CWE
1.1.1	Verifique el uso de un ciclo de vida de desarrollo de software seguro que aborde la seguridad en todas las etapas del desarrollo. (C1)		✓	✓	
1.1.2	Verifique el uso del modelado de amenazas para cada cambio de diseño o planificación de sprint para identificar amenazas, planificar contramedidas, facilitar respuestas de riesgo adecuadas y guiar las pruebas de seguridad.		✓	✓	1053
1.1.3	Verifique que todas las historias y características de usuario contienen restricciones de seguridad funcionales, como por ejemplo: "Como usuario, debería poder ver y editar mi perfil. No debería ser capaz de ver o editar el perfil de nadie más"		✓	✓	1110

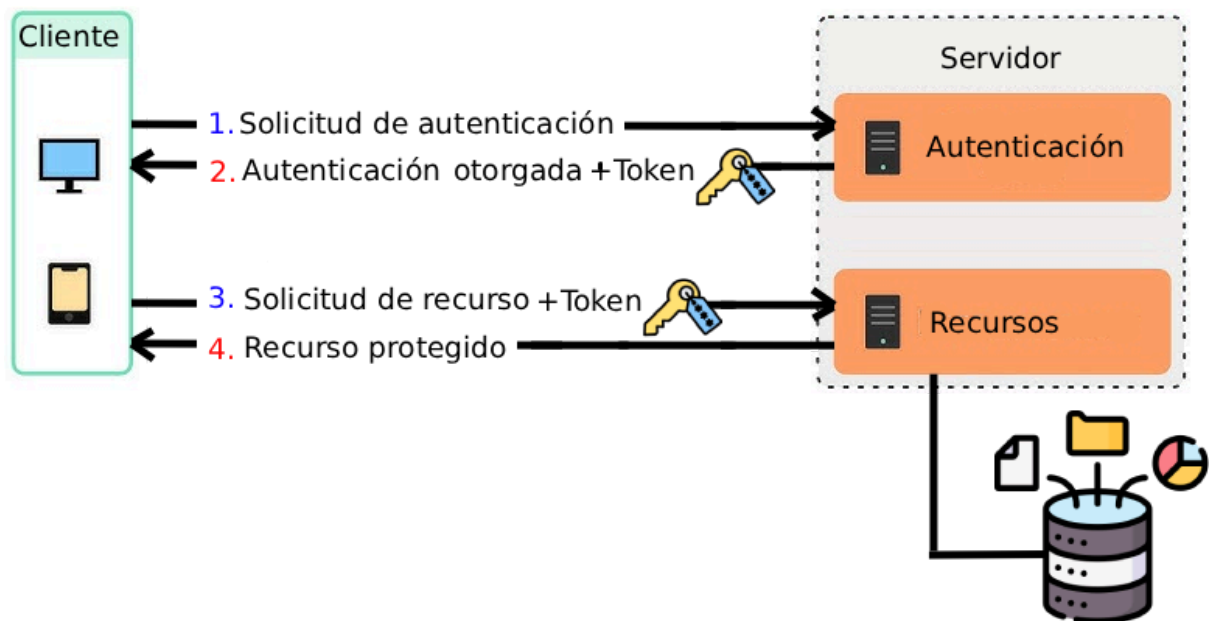
Oauth 2.0

“Open Authorization” (autorización abierta) es un estándar que describe cómo manejar el acceso a un conjunto de recursos, por ejemplo, API remotas.

Proporciona acceso consentido y restringe las acciones que la aplicación del cliente puede realizar en los recursos en nombre del usuario, sin compartir nunca las credenciales del usuario.

Utiliza tokens de acceso (dato que representa la autorización para acceder a los recursos) en nombre del usuario final. Además, por razones de seguridad, los tokens de acceso pueden tener una fecha de caducidad.

[Este ejemplo](#) muestra el funcionamiento del estándar Oauth 2.0.



Hash

Es una función criptográfica que transforma (mediante un algoritmo matemático) cualquier bloque arbitrario de datos en una nueva serie de caracteres con una longitud fija.



MD5

Es un algoritmo de reducción criptográfica que convierte datos en una cadena de 32 caracteres.

Por ejemplo, la palabra “frog” siempre genera este hash `938c2cc0dcc05f2b68c4287040cfcf71`.

Del mismo modo, un dato de 500 caracteres también genera un hash con 32 caracteres. Si cambia un solo carácter del dato de entrada, independientemente de lo grande que sea, la información del hash cambiará completa e irreversiblemente. Nada, salvo una copia exacta, pasará la prueba MD5.

Este algoritmo puede ser utilizado para generar tokens o llaves (keys) únicas para cada usuario, que serán otorgadas al iniciar sesión y solicitadas para verificar su autenticidad en el resto de funciones del servidor.

Guía de GIT y flujo de trabajo centralizado

Containerización

La containerización está transformando el desarrollo y despliegue de software al proporcionar una solución ligera y portátil para empaquetar aplicaciones y sus dependencias con flexibilidad, escalabilidad y eficiencia.

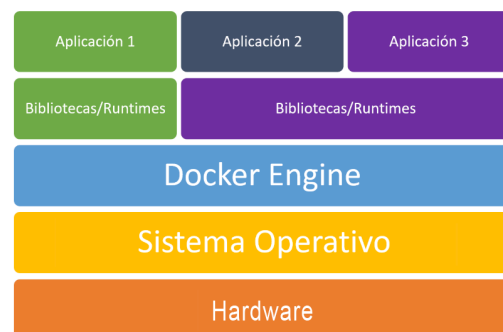
Es una tecnología de virtualización que permite que las aplicaciones se ejecuten en entornos aislados, conocidos como contenedores. Cada contenedor encapsula la aplicación, sus dependencias, bibliotecas y configuraciones, garantizando la consistencia y la reproducibilidad en diferentes entornos.

Contenedores y máquinas virtuales

Para que una máquina virtual pueda ejecutarse es necesario un componente por encima del S.O llamado hipervisor, este software utiliza los recursos de hardware para que otros sistemas operativos creen que se están ejecutando sobre una máquina física. Los hipervisores vienen con productos como Hyper-V (incluido gratuitamente con Windows), VirtualBox o VMWare, entre otros.



Arquitectura de máquinas virtuales



Arquitectura de contenedores

A simple vista solo desaparece la capa del sistema operativo virtualizado, y se sustituye el hipervisor por un componente llamado "Docker Engine".

Sin embargo existe una gran diferencia entre ambas tecnologías, si bien los contenedores tratan también de aislar a las aplicaciones y de generar un entorno replicable y estable para que funcionen, en lugar de albergar un sistema operativo completo lo que hacen es compartir los recursos del propio sistema operativo sobre el que se ejecutan.

Cuando definimos una máquina virtual debemos indicar de antemano cuántos recursos físicos le debemos dedicar. En el caso de los contenedores esto no es así. De hecho no indicamos qué recursos vamos a necesitar, sino que es Docker Engine, en función de las necesidades de cada momento, el encargado de asignar lo que sea necesario para que los contenedores funcionen adecuadamente.

Docker aísla aplicaciones, no sistemas operativos completos, esto hace que los entornos de ejecución de Docker sean mucho más ligeros, y que se aproveche mucho mejor el hardware, además de permitir levantar muchos más contenedores que VMs en la misma máquina física.

Docker

Es una plataforma de containerización que simplifica la creación, distribución y gestión de contenedores. Ofrece una interfaz fácil de usar y un conjunto completo de herramientas para la orquestación de contenedores.

Docker Compose

Es una herramienta que complementa a Docker, permitiendo definir y orquestar múltiples contenedores para crear un entorno de desarrollo completo.

Mientras que Docker se centra en la ejecución y creación de contenedores individuales, Docker Compose se encarga de coordinar y gestionar múltiples contenedores que trabajan juntos para formar una aplicación, esto es especialmente útil cuando se necesita configurar un entorno de desarrollo que involucre múltiples servicios o contenedores. Permite definir la estructura de la aplicación y sus dependencias en un solo archivo YAML, simplificando la creación y gestión del entorno de desarrollo.

docker-compose.yml

Este archivo especifica la versión de Docker Compose, los servicios, las redes y los volúmenes necesarios para la aplicación.

Dentro de la sección servicios podemos definir los diferentes servicios y contenedores que formarán parte del entorno de desarrollo. Cada servicio se define con un nombre único y se especifica su configuración.

Dentro de cada definición de servicio en el archivo docker-compose.yml, se puede incluir configuraciones específicas para ese servicio. Esto permite ajustar parámetros como puertos, volúmenes, variables de entorno y más.

Parámetros y opciones para configurar servicios y contenedores:

- **imagen:** especifica la imagen del contenedor a utilizar.
- **ports:** mapea los puertos del contenedor a puertos de la máquina anfitriona.
- **volúmenes:** monta volúmenes o directorios del sistema de archivos en el contenedor.
- **environment:** define variables de entorno para el contenedor.
- **depends_on:** especifica las dependencias entre servicios.

Plan de testing

Constituye una planificación de las pruebas a realizar sobre un proyecto. En él se establecen los objetivos y alcance de las pruebas, así como también los integrantes del equipo de testing y detalles de las estrategias, criterios y ambiente donde se llevarán a cabo las pruebas.

Ejemplo

 [Plan de Pruebas \(SPACE DIRT\) - ejemplo.pdf](#)

Pruebas de caja negra

Es una metodología de pruebas que se realizan sin conocimiento de los componentes internos de un sistema, se pueden realizar para evaluar la funcionalidad, la seguridad, el rendimiento y otros aspectos de una aplicación.

Pruebas funcionales

Las pruebas funcionales tienen como objetivo validar que una aplicación hace lo que se supone que debe hacer.

Pruebas unitarias

Consisten en aislar una parte del código y comprobar que funciona a la perfección. Son pequeños tests que validan el comportamiento de un objeto y la lógica.

Pruebas de regresión

Se enfocan en verificar que los cambios, mejoras y correcciones de las funciones y las nuevas funciones añadidas no afecten negativamente las funciones existentes o la funcionalidad. Se emplean para asegurar que las funciones existentes sigan respondiendo según lo previsto.

Pruebas no funcionales

Evalúan qué tan bien una aplicación realiza sus funciones principales. Ejemplos de pruebas incluyen pruebas de rendimiento, usabilidad, escalabilidad y seguridad.

Pruebas de caja blanca

Se centran en el código fuente interno de una aplicación, con el objetivo de evaluar la lógica interna del software. Esto implica probar rutas de ejecución, bucles, condiciones y decisiones dentro del código para asegurar su correctitud y eficacia. Son efectivas para identificar errores lógicos, bucles infinitos, divisiones por cero y otros problemas específicos del código.

Lenguaje de programación PHP

PHP es de código abierto y especialmente adecuado para el desarrollo web ya que puede ser incrustado en HTML. El código es ejecutado en el servidor, generando HTML y enviándolo al cliente. El cliente recibirá el resultado de ejecutar el código sin conocerlo.

Hola Mundo en PHP

En informática, "*Hola, mundo*" o "*Hello, World!*" en inglés es un programa que muestra el texto «¡Hola, mundo!» en la mayoría de los casos la pantalla de un monitor. Este programa suele ser usado como introducción al estudio de un lenguaje de programación, siendo un primer ejercicio típico.

```
<html>
  <head>
    <title>Prueba de PHP</title>
  </head>
  <body>
    <?php echo '<p>Hola Mundo</p>'; ?>
  </body>
</html>
```

Elementos y funciones básicas de PHP

Para mostrar una cadena de caracteres podemos utilizar Print o Echo.

Ejemplo print:

```
print ("Hola Mundo");
```

Ejemplo echo:

```
echo ("Hola Mundo");
```

Comentarios

Los comentarios al estilo de "una sola línea" solo comentan hasta el final de la línea o del bloque actual de código de PHP, lo primero que suceda.

```
echo 'Esto es una prueba'; // Esto es un comentario
/* Esto es un comentario multilínea
y otra línea de comentarios */
echo 'Esto es otra prueba';
echo 'Una prueba final'; # Esto es un comentario
```

Variables

En PHP las variables se representan con un signo de pesos (\$) seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas.

Ejemplo:

```
$var1 = 'Roberto';  
$var2 = 'Juan';  
echo ("$var1, $var2"); // imprime "Roberto, Juan"
```

Concatenar

El operador de concatenación '.' retorna el resultado de concatenar sus argumentos derecho e izquierdo.

Ejemplo:

```
$a = "Hello ";  
$b = $a . "World!"; // ahora $b contiene 'Hello World!'
```

Operaciones aritméticas

$\$a + \b	Adición	Suma de \$a y \$b.
$\$a - \b	Sustracción	Diferencia de \$a y \$b.
$\$a * \b	Multipliación	Producto de \$a y \$b.
$\$a / \b	División	Cociente de \$a y \$b.
$\$a \% \b	Módulo	Resto de \$a dividido por \$b.
$\$a ** \b	Exponenciación	Resultado de elevar \$a a la potencia \$b.

Ejemplos:

```
echo (5 + 3); // muestra 8  
echo (-5 - 2); // muestra -7  
echo (5 * 3); // muestra 15  
echo (5 / 2); // muestra 2.5  
echo (5 % 3); // muestra 2
```


Salto de línea

Para producir un salto de línea tenemos dos opciones, podemos utilizar el elemento HTML line break “
” o la función nl2br en combinación con print o echo

Ejemplo con “
”:

```
echo (“<br>”); // Produce un salto de línea en el documento
```

Ejemplo con nl2br:

```
echo nl2br("Hola \n Mundo"); // Produce un salto de línea en el texto
```

Número flotante

Es un número que contiene una parte fraccionaria

Ejemplo:

```
$fl1 = 13.55;  
$fl2 = 5 / 2; // La división retorna 2.5, un valor numérico flotante
```

Redondear un float

La función round() por defecto redondea hacia arriba a partir del valor 0.5.

```
echo round(3.1416); // 3  
echo round(3.5); // 4  
echo round(3.4); // 3
```

Número aleatorio

Genera un número entero aleatorio entre un valor mínimo y uno máximo.

Ejemplo:

```
$dado = rand(1, 6); // retorna un número aleatorio entre 1 y 6  
$moneda = rand(0, 1); // retorna un número aleatorio entre 0 y 1
```

Tipo de dato booleano

Una variable booleana representa dos estados posibles: VERDADERO o FALSO. Los booleanos se utilizan a menudo en pruebas condicionales (que estudiaremos más adelante).

Ejemplo:

```
$x = true;  
$y = false;
```

Arreglo en PHP

Un arreglo es una variable especial que permite almacenar múltiples valores en ella, formando una lista de elementos. Cada elemento es almacenado en una posición del arreglo comenzando por la número 0.

Ejemplo:

```
// Crea un arreglo con nombres de marcas de autos y los lista en pantalla  
$autos = array("Volvo", "BMW", "Toyota");  
echo "Lista de autos: " . $autos[0] . ", " . $autos[1] . " y " . $autos[2] . " .";
```

Para guardar un valor en una posición del arreglo se utiliza la asignación directa (=) especificando el número de la posición:

Ejemplo:

```
$autos[3] = "Renault"; //Asigna una cadena de caracteres en la posición 3 del arreglo  
$autos[4] = "Ford"; //Asigna una cadena de caracteres en la posición 4 del arreglo  
$autos[5] = "Jeep"; //Asigna una cadena de caracteres en la posición 5 del arreglo  
$autos[0] = "Ferrari"; //Sobreescribe el valor de la posición 0 del arreglo
```

Sesiones en PHP

Consiste en una forma de preservar la información del usuario a lo largo de sus accesos subsiguientes. Al acceder al sitio web se le asigna un id único, también llamado id de sesión, éste es almacenado en un archivo dentro del servidor. Cada vez que el mismo ordenador solicita una página con un navegador, también envía una ID que PHP compara con el archivo de sesiones para reanudar o iniciar una nueva.

Registro de variable de sesión

```
<?php
    //crea o reanuda la sesión del cliente
    session_start();

    // verifica si la variable de sesión "count" está definida
    if (!isset($_SESSION['count'])) {
        // asigna un valor a la variable
        $_SESSION['count'] = 0;
    } else {
        // incrementa en 1 el valor de la variable
        $_SESSION['count']++; }
?>
```

Sitio web Multilenguaje

Ejemplo 5: Para crear un sitio web con varios idiomas en PHP, se puede implementar un sistema de gestión de idiomas basado en archivos. Una opción sencilla es utilizar **arreglos** (arrays) y **sesiones** (\$_SESSION) PHP para almacenar el idioma seleccionado.

Cada archivo PHP contiene un conjunto de textos traducidos en un idioma (ej. es.php, en.php, fr.php).

Una función carga los textos traducidos según el idioma seleccionado y los muestra en la página principal.

El idioma seleccionado por el usuario es guardado en una variable de sesión para ser utilizado a través de las diferentes páginas.

Ejemplo 6: Otro enfoque es utilizar una **API** que almacene el contenido en diferentes idiomas y lo entregue en JSON. Luego utilizar un script **JavaScript** para cargar y mostrar el contenido correcto basado en la selección del usuario.

Servidor web interno de PHP

Un servidor es un sistema que proporciona recursos, datos, servicios o programas a otros ordenadores, conocidos como clientes, a través de una red.

Cuando un servidor se administra desde la línea de comandos de otro servidor, el servidor administrado se denomina remoto, pero si se administra desde su propia línea de comandos, se denomina servidor local.

El servidor web de PHP ha sido diseñado para ayudar al desarrollo de aplicaciones. Este ejecuta solamente un único proceso, por lo que las aplicaciones PHP se detendrán si la solicitud está bloqueada.

La raíz de la ejecución es el directorio actual de trabajo donde se inició el servidor interno y si no se especifica un archivo, entonces se buscará ejecutar el archivo index.php o .html que estén en el directorio

Este servidor también puede ser útil para realizar pruebas o demostraciones en entornos controlados. No se pretende que sea un servidor web con todas las funciones y por lo tanto no debe ser utilizado en una red pública.

Comando para iniciar servidor interno en la ubicación actual

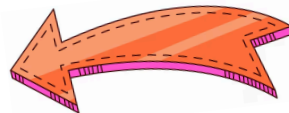
```
php -S localhost:8000
```

Comando para iniciar servidor interno en un directorio específico

```
php -S localhost:8000 -t pruebas/
```

Material teórico, ejemplos y ejercicios de Bases de datos

bit.ly/materialbdatos



Aplicaciones de ejemplo

1. Subir archivo a servidor con PHP. [Enlace](#)

2. Login y logout de usuario con PHP (proyecto de ejemplo) [Enlace](#)

3. Conexión y consulta a base de datos MySQL

Base de datos:

```
CREATE DATABASE IF NOT EXISTS base_usuarios;  
CREATE TABLE IF NOT EXISTS base_usuarios.usuario (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  usr_name VARCHAR(100) NOT NULL,  
  usr_email VARCHAR(100) UNIQUE NOT NULL,  
  usr_pass VARCHAR(100) NOT NULL,  
  imagen VARCHAR(100) DEFAULT NULL,  
  PRIMARY KEY (id)  
);  
INSERT INTO base_usuarios.usuario(usr_name, usr_email, usr_pass) VALUES  
('Usuario1','usuario122@gmail.com','123456');
```

```
<?php  
  // Database configuration  
  $hostname = 'localhost';  
  $username = 'root';  
  $password = '';  
  $database = 'base_usuarios';  
  // Establish database connection  
  $conn = mysqli_connect($hostname, $username, $password, $database);  
  // Check connection  
  if(!$conn){  
      die('Connection failed: ' . mysqli_connect_error());  
  }  
  
  // Make query  
  $query = "SELECT * FROM usuario";  
  $result = mysqli_query($conn, $query);  
  $usuarios = [];  
  while($row = mysqli_fetch_assoc($result)) {  
      $usuarios[] = $row;  
  }  
  print_r($usuarios);  
?>
```

4. API Restful y Solicitudes desde JavaScript [Enlace](#)

- + Puedes encontrar la teoría utilizada en este ejemplo en el siguiente [enlace](#).
 - + Este ejemplo utiliza [Bootstrap](#), [JavaScript](#) y POO con PHP.
 - + La [base de datos](#) utilizada corresponde a la del [ejemplo 3](#).
-

5. Sitio web Multilenguaje

- + Puedes encontrar la teoría utilizada en estos ejemplos en el siguiente [enlace](#).
- + Estos ejemplos utilizan [Bootstrap](#) y [JavaScript](#).

a. Desde archivos PHP [Enlace](#)

b. Desde API y con Javascript [Enlace](#)

6. Seguridad: Login con API (Token/KEY Oauth 2.0) [Enlace](#)

- + Puedes encontrar la teoría utilizada en este ejemplo en el siguiente [enlace](#).
 - + Este ejemplo utiliza [Bootstrap](#), [JavaScript](#) y POO con PHP.
 - + Base de datos utilizada

```
CREATE DATABASE IF NOT EXISTS base_usuarios;  
CREATE TABLE IF NOT EXISTS base_usuarios.usuario (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  usr_name VARCHAR(100) NOT NULL,  
  usr_email VARCHAR(100) UNIQUE NOT NULL,  
  usr_pass VARCHAR(100) NOT NULL,  
  imagen VARCHAR(100) DEFAULT NULL,  
  PRIMARY KEY (id)  
);  
CREATE TABLE IF NOT EXISTS base_usuarios.access_token (  
  token CHAR(32) NOT NULL,  
  id_usuario INT(11) NOT NULL,  
  fecha_creado DATETIME NOT NULL DEFAULT NOW(),  
  fecha_vencimiento DATETIME NOT NULL DEFAULT (NOW() + INTERVAL 12 HOUR),  
  PRIMARY KEY (token),  
  FOREIGN KEY (id_usuario) REFERENCES usuario(id) ON DELETE CASCADE  
);
```
-

7. Testing: Pruebas automatizadas de funcionalidades de API [Enlace](#)

- + La base de datos utilizada corresponde a la del [ejemplo 6](#).
 - + Este ejemplo utiliza [Bootstrap](#), [JavaScript](#) y POO con PHP.
-

8. API Restful y Solicitudes desde videojuego en JavaScript [Enlace](#)

- + Este ejemplo utiliza [Bootstrap](#), [JavaScript](#) y POO con PHP.
- + Base de datos utilizada

```
CREATE DATABASE IF NOT EXISTS base_usuarios;
CREATE TABLE IF NOT EXISTS base_usuarios.usuario (
  id INT(11) NOT NULL AUTO_INCREMENT,
  usr_name VARCHAR(100) NOT NULL,
  usr_email VARCHAR(100) UNIQUE NOT NULL,
  usr_pass VARCHAR(100) NOT NULL,
  imagen VARCHAR(100) DEFAULT NULL,
  PRIMARY KEY (id)
);
CREATE TABLE IF NOT EXISTS base_usuarios.access_token (
  token CHAR(32) NOT NULL,
  id_usuario INT(11) NOT NULL,
  fecha_creado DATETIME NOT NULL DEFAULT NOW(),
  fecha_vencimiento DATETIME NOT NULL DEFAULT (NOW() + INTERVAL 12 HOUR),
  PRIMARY KEY (token),
  FOREIGN KEY (id_usuario) REFERENCES usuario(id) ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS base_usuarios.partida (
  id INT(11) NOT NULL AUTO_INCREMENT,
  id_usuario INT(11) NOT NULL,
  nombre_oponente VARCHAR(100) NOT NULL,
  fecha DATETIME NOT NULL DEFAULT NOW(),
  movimientos VARCHAR(50) NOT NULL DEFAULT "O,,,,,,,,",
  resultado INT(1) DEFAULT 0, /* 0: en curso, 1: ganado, 2: perdido, 3: empate */
  PRIMARY KEY (id),
  FOREIGN KEY (id_usuario) REFERENCES usuario(id) ON DELETE CASCADE
);
```

9. Seguridad: Ataque XSS. [Enlace](#)

- + Este ejemplo muestra una vulnerabilidad documentada en el standar
- + Utiliza [Bootstrap](#) para su interfaz de usuario.

10. Seguridad: Inyección SQL. [Enlace](#)

- + Este ejemplo muestra una vulnerabilidad documentada en el standar
- + Utiliza [Bootstrap](#) para su interfaz de usuario.
- + [Base de datos](#) utilizada

```
CREATE DATABASE IF NOT EXISTS base_usuarios;  
CREATE TABLE IF NOT EXISTS base_usuarios.usuario (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  usr_name VARCHAR(100) NOT NULL,  
  usr_email VARCHAR(100) UNIQUE NOT NULL,  
  usr_pass VARCHAR(100) NOT NULL,  
  imagen VARCHAR(100) DEFAULT NULL,  
  PRIMARY KEY (id)  
);
```

X. Seguridad: Sesiones y cookies.

https://drive.google.com/file/d/15nUdpCsN_VfIHUVwMoxNdMntve5XEP6H/view?usp=sharing

X. (GET) Consumir API restful

<https://academy.leewayweb.com/como-consumir-un-webservice-rest-con-php/>

X. Crear API restful

<https://drive.google.com/file/d/1br8wVX6oyL6wPt9OQ6sHYmIGmK5VvatE/view?usp=sharing>

base de datos::

```
CREATE DATABASE base_usuarios;  
CREATE TABLE base_usuarios.usuarios (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  usr_name VARCHAR(100) NOT NULL,  
  usr_email VARCHAR(100) UNIQUE NOT NULL,  
  usr_pass VARCHAR(100) NOT NULL,  
  imagen VARCHAR(100),  
  PRIMARY KEY (id)  
);  
INSERT INTO base_usuarios.usuarios(usr_name, usr_email, usr_pass) VALUES  
('User1','user122@gmail.com','123456');
```

Fuente: <https://wpwebinfotech.com/blog/how-to-build-simple-rest-api-in-php/>

X. (POST) Consumir API restful desde formulario html

Los navegadores no soportan JSON como un *media type* para el envío de datos a través de un *form*, una manera de hacerlo es utilizando JavaScript y la librería *XMLHttpRequest* de la siguiente forma:

Proyecto de ejemplo: [Crear API Restful](#)

```
index.php X
index.php
117 <hr>
118 <h4>Crear usuario (POST)</h4>
119 <div>
120   <label for="usr_name">Nombre:</label>
121   <input type="text" name="usr_name" id="usr_name">
122   <br>
123   <label for="usr_age">Edad:</label>
124   <input type="number" name="usr_age" id="usr_age">
125   <br>
126   <label for="usr_email">Email:</label>
127   <input type="text" name="usr_email" id="usr_email">
128   <br>
129   <label for="usr_pass">Contraseña:</label>
130   <input type="password" name="usr_pass" id="usr_pass">
131   <br>
132   <button type="submit" onclick="crearUsuario();">Crear</button>
133 </div>
134
135 <script>
136   function crearUsuario(){
137     usr_name = document.getElementById("usr_name").value;
138     usr_age = document.getElementById("usr_age").value;
139     usr_email = document.getElementById("usr_email").value;
140     usr_pass = document.getElementById("usr_pass").value;
141
142     var xmlhttp = new XMLHttpRequest(); // new HttpRequest instance
143     var theUrl = "http://localhost/API_restful/api.php/usuarios";
144     xmlhttp.open("POST", theUrl);
145     xmlhttp.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
146     xmlhttp.send(JSON.stringify({
147       "usr_name": usr_name,
148       "usr_age": usr_age,
149       "usr_email": usr_email,
150       "usr_pass": usr_pass
151     }));
152     location.reload();
153   }
154 </script>
155 </body>
156 </html>
```

Añadir al archivo index.php un formulario para completar con datos del nuevo usuario.

- Al presionar el botón "Crear" ejecuta una función JavaScript
- La función *crearUsuario()* obtiene los datos ingresados al formulario y efectúa el envío de un objeto JSON hacia el endpoint de la API.
- Luego reinicia la página con *location.reload()*.

X. (DELETE) Consumir API restful desde formulario html

Proyecto de ejemplo: [Crear API Restful](#)

```
index.php
133 <hr>
134 <div>
135     <h4>Eliminar usuario</h4>
136     <label for="usr_id">ID usuario:</label>
137     <input type="number" name="usr_id" id="usr_id">
138     <br>
139     <button type="submit" onclick="eliminarUsuario();">Eliminar</button>
140 </div>
141
142 <script>
143     function eliminarUsuario(){
144         usr_id = document.getElementById("usr_id").value;
145
146         var xmlhttp = new XMLHttpRequest();
147         var url = "http://localhost/API_restful/api.php/usuarios/"+usr_id;
148         xmlhttp.open("DELETE", url);
149         xmlhttp.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
150         xmlhttp.send(JSON.stringify({
151             "usr_id": usr_id,
152         }));
153         location.reload();
154     }
155 </script>
156 </body>
157 </html>
```

Añadir al archivo index.php un formulario para completar con el id del usuario a eliminar.

- Al presionar el botón "Eliminar" ejecuta una función JavaScript
- La función *eliminarUsuario()* obtiene el ID ingresado al formulario y efectúa el envío de un objeto JSON hacia el endpoint de la API.
- Luego reinicia la página con *location.reload()*.

X. (PUT) Consumir API restful desde formulario html

Proyecto de ejemplo: [Crear API Restful](#)

```
index.php
117 <hr>
118 <h4>Modificar usuario (PUT)</h4>
119 <div>
120     <label for="usr_id">ID:</label>
121     <input type="number" name="usr_id" id="usr_id">
122     <br>
123     <label for="usr_name">Nombre:</label>
124     <input type="text" name="usr_name" id="usr_name">
125     <br>
126     <label for="usr_age">Edad:</label>
127     <input type="number" name="usr_age" id="usr_age">
128     <br>
129     <label for="usr_email">Email:</label>
130     <input type="text" name="usr_email" id="usr_email">
131     <br>
132     <label for="usr_pass">Contraseña:</label>
133     <input type="password" name="usr_pass" id="usr_pass">
134     <br>
135     <button type="submit" onclick="modificarUsuario();">Modificar</button>
136 </div>
137
138 <script>
139     function modificarUsuario(){
140         usr_id = document.getElementById("usr_id").value;
141         usr_name = document.getElementById("usr_name").value;
142         usr_age = document.getElementById("usr_age").value;
143         usr_email = document.getElementById("usr_email").value;
144         usr_pass = document.getElementById("usr_pass").value;
145
146         var xmlhttp = new XMLHttpRequest(); // new HttpRequest instance
147         var theUrl = "http://localhost/programasphp/API_restful/api.php/usuarios/"+usr_id;
148         xmlhttp.open("PUT", theUrl);
149         xmlhttp.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
150         xmlhttp.send(JSON.stringify({
151             "usr_name": usr_name,
152             "usr_age": usr_age,
153             "usr_email": usr_email,
154             "usr_pass": usr_pass
155         }));
156         location.reload();
157     }
158 </script>
159 </body>
160 </html>
```

Añadir al archivo index.php un formulario para completar con el id y los datos del usuario a modificar.

- Al presionar el botón “Modificar” ejecuta una función JavaScript
- La función *modificarUsuario()* obtiene el ID ingresado al formulario y efectúa el envío de un objeto JSON hacia el endpoint de la API.
- Luego reinicia la página con *location.reload()*.

```
api.php 16 switch ($method) {
    case 'PUT':
        if (preg_mat
            // Updat
            $usuario:
            // $data
            parse_sti
            $result :
            echo jsoi
        }
        break;
    case 'DELETE':
        47

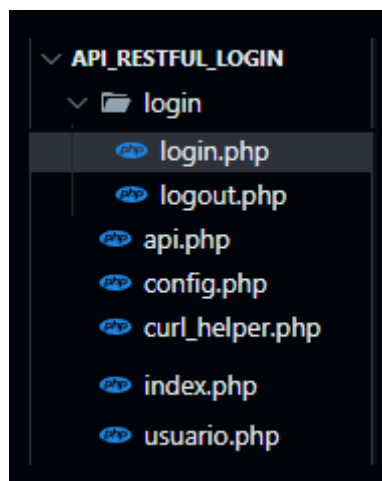
api.php 16 switch ($method) {
    case 'PUT':
        if (preg_match('/^\/usuarios\/(\d+)\/$', $endpoint, $matches)) {
            // Update usuario bi ID
            $usuarioId = $matches[1];
            $data = json_decode(file_get_contents('php://input'), true);
            //parse_str(file_get_contents('php://input'), $data);
            $result = $usuarioObj->updateUsuario($usuarioId, $data);
            echo json_encode(['success' => $result]);
        }
        break;
    case 'DELETE':
        47
```

Cambia la línea comentada en el archivo *api.php* que obtiene los datos de la solicitud, esta se encuentra en el caso *PUT* del switch que evalúa el verbo de la solicitud.

11. Login de usuario a través de API

Proyectos de ejemplo: [Crear API Restful](#) y [Login y logout de usuario](#)

En este ejemplo se combinan los proyectos de ejemplo para formar uno solo que contiene el inicio de sesión (asignación y comprobación de variables de sesión) y el uso de API.



El archivo login.php contiene una verificación de variable de sesión, en caso de encontrarse una definida, se redirecciona al usuario al index.php (no tiene sentido mostrar el formulario de inicio de sesión cuando esta ya fue iniciada).

El método del formulario es POST (para no mostrar la contraseña en la url) y envía los datos directamente a la API hacia el endpoint “/login” (api.php/login).

login.php

```
<?php
    session_start();
    if(isset($_SESSION['usr_email'])){
        header('location: ../index.php');
    }
?>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Login</title>
</head>
<body>
    <h4>Login de usuario (POST)</h4>
    <form action="../api.php/login" method="POST">
        <label for="usr_email">Email: </label>
        <input type="text" name="usr_email" id="usr_email">
        <br>
        <label for="usr_pass">Contraseña: </label>
        <input type="password" name="usr_pass" id="usr_pass">
        <br>
        <button type="submit">Login</button>
    </form>

    <script>
        // comprueba si la url contiene un mensaje (msg) para mostrar
        const searchParams = new
URLSearchParams(window.location.search);
        if(searchParams.has("msg")){
            alert(searchParams.get("msg"));
        }
    </script>
</body>
</html>
```

En el archivo api.php añade el “endpoint” /login, este ejecutará el método del usuario llamado “loginUsuario()”

api.php


```
case 'POST':
    if($endpoint === '/usuarios'){
        // Add new usuario
        $data = json_decode(file_get_contents('php://input'), true);
        $result = $usuarioObj->addUsuario($data);
        echo json_encode(['success' => $result]);
    }else if($endpoint === '/login'){
        // loguea al usuario
        $data = json_decode(file_get_contents('php://input'), true);
        $result = $usuarioObj->loginUsuario($data);
        //echo $result;
        echo json_encode(['success' => $result]);
    }
}
```

Añade a la clase usuario el método loginUsuario(), éste obtiene el email y contraseña del formulario y busca el usuario correspondiente en la base de datos, si obtiene un resultado guarda su email en una variable de sesión y lo redirecciona a index.php. En el caso que la consulta no devuelve un resultado redirecciona al usuario al archivo login.php con un mensaje en la url.

usuario.php

```
public function loginUsuario($data)
{
    $usr_email = $_POST['usr_email'];
    $usr_pass = $_POST['usr_pass'];
    $query = "SELECT * FROM usuarios WHERE usr_email = '$usr_email'
and usr_pass = $usr_pass";
    $result = mysqli_query($this->conn, $query);
    $usuario = mysqli_fetch_assoc($result);
    if($usuario){
        $_SESSION['usr_email'] = $usr_email;
        header('location: ../index.php');
    }else{
        header('location: ../login/login.php?msg=Email o contraseña
incorrectos');
    }
}
```

Por último en el archivo index.php se añade una verificación de la variable de sesión, en caso de no estar definida, redirecciona al usuario al archivo login.php con un mensaje en la url.

 index.php

```
<?php
    // Include the common cURL file
    require_once 'curl_helper.php';

    // si el usuario no inició sesión, lo redirige al login
    session_start();
    if(!isset($_SESSION['usr_email'])){
        header('location: login/login.php?msg=Debes iniciar sesion');
    }
?>
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
<h4>Perfil del usuario: </h4>
<h3><?php echo "bienvenido ".$_SESSION['usr_email'] ?></h3>
<a href="login/logout.php">Logout</a>
</body>
</html>
```

12. (Oauth 2.0) Login de usuario a través de API con token

Proyecto que toma el [ejemplo anterior](#) para implementar un login de usuario donde el cliente obtiene un token y lo utiliza para ser autorizado a crear otros usuarios dentro de la API.

base de datos::


```
CREATE DATABASE base_usuarios;
CREATE TABLE base_usuarios.usuarios (
  id INT(11) NOT NULL AUTO_INCREMENT,
  usr_name VARCHAR(100) NOT NULL,
  usr_age INT(3) NOT NULL,
  usr_email VARCHAR(100) NOT NULL,
  usr_pass VARCHAR(100) NOT NULL,
  PRIMARY KEY (id)
);
```

Descarga el ejemplo:

https://drive.google.com/file/d/1rLKCVI3tMoT0BfYSMxn-NF3DmTHipHNC/view?usp=drive_link

El token es generado utilizando el algoritmo de hashing *MD5* utilizando el email y la contraseña del usuario, y un dato de que el usuario no conoce: una cadena de caracteres con la palabra “sabor” para añadir dificultad en caso que alguien desee romper esta medida de seguridad.

En caso de un inicio de sesión exitoso el usuario es redirigido al index, en caso contrario el usuario es llevado al formulario de inicio de sesión con un mensaje para mostrar.

 usuario.php X

```
private $salt = "sabor";

public function loginUsuario($usr_email, $usr_pass)
{
    $query = "SELECT * FROM usuarios WHERE usr_email = '$usr_email' and usr_pass = '$usr_pass'";
    $result = mysqli_query($this->conn, $query);
    $usuario = mysqli_fetch_assoc($result);
    if($usuario){
        $usr_key = md5($usr_email.$usr_pass.$salt);
        $_SESSION['usr_key'] = $usr_key;
        $_SESSION['usr_email'] = $usr_email;
        header('location: ../index.php');
    }else{
        header('location: ../login/login.php?msg=Email o contraseña incorrectos');
    }
}
```


El formulario de inicio de sesión envía los datos a la api al endpoint `/login` mediante el método `POST`. El script busca en la url el parámetro `msg` que corresponde a un mensaje para mostrar al usuario.

login.php X

```
<body>
  <h2>Login de usuario</h2>
  <form action="../../api.php/login" method="POST">
    <label for="usr_email">Email: </label>
    <input type="text" name="usr_email" id="usr_email">
    <br>
    <label for="usr_pass">Contraseña: </label>
    <input type="password" name="usr_pass" id="usr_pass">
    <br>
    <button type="submit">Login</button>
  </form>

  <script>
    // comprueba si la url contiene un mensaje (msg) para mostrar
    const searchParams = new URLSearchParams(window.location.search);
    if(searchParams.has("msg")){
      alert(searchParams.get("msg"));
    }
  </script>
</body>
```

En el index se encuentra el formulario para crear otros usuarios, solo es posible acceder a este si el usuario cuenta con las variables de sesión correspondientes, de lo contrario será redirigido al inicio de sesión.

Se da la bienvenida al usuario y se muestra el token que obtuvo del servidor al iniciar la sesión (el token de inicio de sesión es revelado para demostración, no es recomendable hacerlo público).

Mediante los inputs de tipo *hidden* el formulario de registro esconde datos del usuario (su token y correo electrónico) en su interior, estos datos serán utilizados por la API para verificar si el usuario está autorizado para hacer uso de la misma.

Al presionar el botón *Registrar* el formulario envía los datos mediante el método *POST* hacia el endpoint */usuarios* de la API.



index.php X

```
<?php
    session_start();
    if(!isset($_SESSION['usr_email']) || !isset($_SESSION['usr_key'])){
        header('location: login/login.php');
    }
?>
<!DOCTYPE html>

<body>
    <h3><?php echo "bienvenido " . $_SESSION['usr_email']; ?></h3>
    <p><?php echo "key: " . $_SESSION['usr_key']; ?></p>
    <h2>Registrar nuevo usuario</h2>
    <form action="api.php/usuarios" method="POST">
        <label for="email">Email: </label>
        <input type="text" name="email" id="email">
        <br>
        <label for="pass">Contraseña: </label>
        <input type="password" name="pass" id="pass">
        <br>
        <input type="hidden" name="usr_key" value="<?php echo $_SESSION['usr_key']; ?>">
        <input type="hidden" name="usr_email" value="<?php echo $_SESSION['usr_email']; ?>">
        <button type="submit">Registrar</button>
    </form>
    <a href="login/logout.php">Cerrar sesion</a>

    <script>
        // comprueba si la url contiene un mensaje (msg) para mostrar
        const searchParams = new URLSearchParams(window.location.search);
        if(searchParams.has("msg")){
            alert(searchParams.get("msg"));
        }
    </script>
</body>
```

Finalmente la API recibe los datos del formulario de registro y verifica si el token es válido, la verificación consiste en reproducir el token del usuario y compararlo con el que proporcionó, si ambos son iguales entonces el usuario es autorizado a registrar los datos en la API. En caso que los tokens sean distintos no se efectúa el registro, se destruye la sesión del usuario y se lo redirecciona al formulario de inicio de sesión, posiblemente se trate de un caso de falsificación de credenciales.

 usuario.php 

```
public function addUsuario($email, $pass, $usr_email, $usr_key)
{
    // evaluar la key
    $query0 = "SELECT * FROM usuarios WHERE usr_email = '$usr_email'";
    $result = mysqli_query($this->conn, $query0);
    $usuario = mysqli_fetch_assoc($result);
    $server_key = md5($usuario["usr_email"].$usuario["usr_pass"].$salt);
    if(strcmp($usr_key,$server_key)==0){
        $query = "INSERT INTO usuarios (usr_name, usr_age, usr_email, usr_pass) VALUES ('Nombre', '0', '$email', '$pass')";
        $result = mysqli_query($this->conn, $query);
        $msg = "Usuario registrado.";
        header('location: ../index.php?msg='.$msg);
    }else{
        $msg = "Error: credenciales incorrectas.";
        $_SESSION = array();
        session_destroy();
        header('location: ../login/login.php?msg='.$msg);
    }
}
```

13. (Docker) Despliegue de aplicación con contenedores

Proyecto que toma el [ejemplo anterior](#) y crea un contenedor con las imágenes de Apache, MySQL, PHP y la aplicación desarrollada.

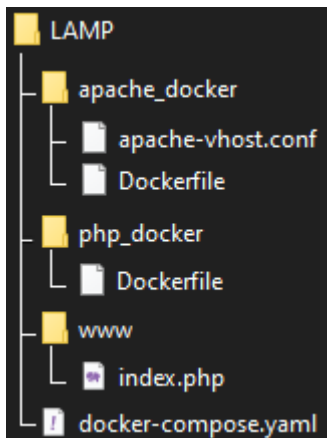
- [¿Qué es un contenedor?](#)

Resumen:

1. Crear la estructura para almacenar los archivos requeridos.
2. Instalar Docker y Docker Compose.
3. Desde la ubicación del archivo `docker-compose.yaml` ejecutar el comando `docker-compose up -d`
4. Crear las tablas de la base de datos en el contenedor `mysql`
5. Editar el `hostname` en el archivo `config.php`

1. Crear la estructura para almacenar los archivos requeridos.

Estos archivos incluyen las instrucciones para crear los servicios de Apache, MySQL y PHP. Dentro del directorio `www` se encuentra el proyecto.



Contenido del archivo LAMP/apache_docker/**apache-vhost.conf**

```
ServerName localhost

LoadModule deflate_module /usr/local/apache2/modules/mod_deflate.so
LoadModule proxy_module /usr/local/apache2/modules/mod_proxy.so
LoadModule proxy_fcgi_module /usr/local/apache2/modules/mod_proxy_fcgi.so

<VirtualHost *:80>
    # Proxy .php requests to port 9000 of the PHP container
    ProxyPassMatch ^/(.*\.php(/.*)?)$ fcgi://php:9000/var/www/html/$1
    DocumentRoot /var/www/html/
    <Directory /var/www/html/>
        DirectoryIndex index.php
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    # Send Apache logs to stdout and stderr
    CustomLog /proc/self/fd/1 common
    ErrorLog /proc/self/fd/2
</VirtualHost>
```

Contenido del archivo LAMP/apache_docker/**Dockerfile**

```
FROM httpd:2.4-alpine

#RUN apk update; apk upgrade;

# Copy Apache virtual host file to proxy .php requests to PHP container
COPY apache-vhost.conf /usr/local/apache2/conf/apache-vhost.conf
RUN echo "Include /usr/local/apache2/conf/apache-vhost.conf" \
    >> /usr/local/apache2/conf/httpd.conf
```

Contenido del archivo LAMP/php_docker/**Dockerfile**

```
FROM php:8.2-fpm

# Installing dependencies for building the PHP modules
RUN apt update && \
    apt install -y zip libzip-dev libpng-dev libicu-dev libxml2-dev

# Installing additional PHP modules
RUN docker-php-ext-install mysqli pdo pdo_mysql gd zip intl xml

# Cleaning APT cache
RUN apt clean
```

Contenido del archivo LAMP/www/index.php (ejemplo)

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

Contenido del archivo LAMP/docker-compose.yml

```
services:

  # PHP Service
  php:
    build: './php_docker/'
    volumes:
      - ./www:/var/www/html/

  # Apache Service
  apache:
    build: './apache_docker/'
    depends_on:
      - php
    ports:
      - "80:80"
    volumes:
      - ./www:/var/www/html/

  # MariaDB Service
  mariadb:
    environment:
      MYSQL_ALLOW_EMPTY_PASSWORD: true
      MYSQL_DATABASE: base_usuarios
    container_name: mysql
    image: mysql:8.0
    volumes:
      - mysqldata:/var/lib/mysql
    #Map port 3306 on the mysql container to port 3306 in the host
    ports:
      - 3306:3306

# Volumes
volumes:
  mysqldata:
```

2. Instalar Docker

- + Windows (Docker desktop): <https://docs.docker.com/desktop/install/windows-install/>
- + RockyLinux (RHEL): <https://docs.docker.com/engine/install/rhel/#set-up-the-repository>

Instalar Docker Compose

- + RockyLinux (RHEL): <https://docs.docker.com/compose/install/linux/>

Docker Desktop (para Windows) instala Docker engine y utiliza una interfaz CLI para la gestión de los contenedores

3. Desde la ubicación del archivo *docker-compose.yml* ejecutar el comando:

```
docker-compose up -d
```

Este comando construye, (re)crea, inicia y adjunta contenedores y servicios.










```
C:\xampp\htdocs\API_restful\LAMP>docker-compose up
[+] Running 5/5
 ✓ Network lamp_default      Created
 ✓ Volume "lamp_mysqldata"   Created
 ✓ Container lamp-php-1      Created
 ✓ Container mysql           Created
 ✓ Container lamp-apache-1   Created
Attaching to apache-1, php-1, mysql
```

Desde la aplicación Docker desktop se pueden visualizar las imágenes y contenedores.

docker desktop PERSONAL	
Containers	
Images	
Volumes	
Builds	
Docker Scout	
Extensions	





Name	Tag
mysql bf79508626d6	8.0
lamp-php a3f064eac996	latest
lamp-apache f00e97464f1d	latest

Cuando un contenedor está en ejecución lo hace a través de un puerto específico.

Name	Image	Status	Port(s)
 lamp		Running (3/3)	
 php-1 fceb9b8ed855 	lamp-php:<none>	Running	
 mysql c0f95818b756 	mysql:8.0	Running	3306:3306 
 apache-1 04220203e60a 	lamp-apache:<none>	Running	80:80 

4. Crear las tablas de la base de datos en el contenedor *mysql*

Containers / mysql


 c0f95818b756  [mysql:8.0](#)
[3306:3306](#) 

[Logs](#)
[Inspect](#)
[Bind mounts](#)
[Exec](#)
[Files](#)
[Stats](#)

```

sh-5.1# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.40 MySQL Community Server - GPL

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE IF NOT EXISTS base_usuarios;
Query OK, 1 row affected, 1 warning (0.04 sec)

mysql> CREATE TABLE base_usuarios.usuarios (
  id INT(11) NOT NULL AUTO INCREMENT

```


Base de datos de ejemplo para este proyecto:

```
CREATE DATABASE IF NOT EXISTS base_usuarios;  
CREATE TABLE IF NOT EXISTS base_usuarios.usuario (  
  id INT(11) NOT NULL AUTO_INCREMENT,  
  usr_name VARCHAR(100) NOT NULL,  
  usr_email VARCHAR(100) UNIQUE NOT NULL,  
  usr_pass VARCHAR(100) NOT NULL,  
  imagen VARCHAR(100) DEFAULT NULL,  
  PRIMARY KEY (id)  
);  
INSERT INTO base_usuarios.usuario(usr_name, usr_email, usr_pass) VALUES  
('Usuario1','usuario122@gmail.com','123456');
```

5. Editar el *hostname* en el archivo *config.php*

De esta manera la aplicación apunta al contenedor llamado mysql que contiene instalado servidor de bases de datos MySQL.

Contenido del archivo *config.php*

```
<?php  
// Database configuration  
$hostname = 'mysql';  
$username = 'root';  
$password = '';  
$database = 'base_usuarios';  
// Establish database connection  
$conn = mysqli_connect($hostname, $username, $password, $database);  
// Check connection  
if(!$conn){  
    die('Connection failed: ' . mysqli_connect_error());  
}  
?>
```

14. Lectura y escritura de archivo de texto

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=598:write-php-lectura-y-escritura-fichero-txt-modo-acceso-phpeol-salto-de-linea-ejercicio-ejemplo-cu00837b&catid=70&Itemid=193

15. Verificación de existencia y creación de directorio en el servidor

```
<?php
    // Verifica existencia de directorio
    if(is_dir('directorio')){
        echo "El directorio ya existe. <br>";
    }else{
        // Crea directorio
        mkdir("directorio", 0777);
        echo "Directorio creado. <br>";
    }
?>
```