UD 1. Arquitecturas y tecnologías de programación Web en entorno servidor

Índice de contenidos

1. Introducción

En esta unidad estudiaremos cómo se producen las comunicaciones web entre usuarios y servidores y por tanto qué ocurre desde que introducimos la dirección URL de una web en el navegador hasta que este nos la muestra en pantalla: lenguajes involucrados, software, protocolo, arquitectura...

2. Modelos de programación en entornos cliente/servidor

Una de las arquitecturas de software más utilizadas en informática es la llamada **arquitectura cliente / servidor**. En este tipo de arquitectura hay aplicaciones que hacen peticiones (el **cliente**) y otra aplicación que está a la espera de recibir esas peticiones y, cuando las recibe, las procesa y responde (el **servidor**). Ejemplos de esta arquitectura son los servidores **NTP** (Network Time Protocol), a los cuales se le pide la hora actual para sincronizarse, los servidores **FTP** (File Transfer Protocol), a los que se les puede enviar y pedir archivos, o precisamente los servidores **HTTP**, que son los encargados de servir los recursos web.

2.1. Protocolo HTTP

El **protocolo HTTP** (Hypertext Transfer Protocol, Protocolo de Transferencia de Hipertexto), es el protocolo de comunicación en red que especifica cómo deben comunicarse el cliente y el servidor. Es el protocolo sobre el que está construida la World Wide Web, la web tal y como la conocemos. Está pensado para servir, entre otros, archivos **HTML**, pero como los documentos web incluyen archivos CSS, JavaScript, imágenes, sonidos... se puede servir **cualquier tipo de archivo**. En general se habla de recurso (de ahí el término URL que se describe más abajo) y no de archivo.

- En este protocolo los clientes suelen usar los **navegadores**, aplicaciones que están instalados en los ordenadores, móviles, tablets... de los usuarios que a través de un URL solicitan a un servidor un recurso web y, cuando la reciben, la muestran al usuario.
- Los servidores **HTTP** son las aplicaciones encargadas de atender las peticiones de los navegadores y ofrecer los recursos web de los que dispone. Utilizan por defecto el **puerto 80** para atender peticiones, pero es posible utilizar cualquier otro (comúnmente 8080, o 8888 o 8000). Aunque podemos instalar un servidor HTTP en cualquier equipo, en el ámbito profesional lo común es trabajar con alguna compañía de **hosting web** (alojamiento de páginas web). Estas compañías suelen disponer de servidores (equipos muy potentes especializados en responder muchas peticiones) en los que alojar los recursos web. La gama de servicios va desde simplemente alojar la web hasta ofrecer control total sobre el servidor contratado.

El protocolo **HTTPS** (Hypertext Transfer Protocol Secure, Protocolo de Transferencia de Hipertexto Seguro) es una variante de HTTP en la que la comunicación va cifrada para mayor

seguridad. Los servidores HTTPS suelen utilizar el **puerto 443**. Tradicionalmente se utilizaba únicamente cuando se enviaba información sensible (contraseñas, pagos online...) pero actualmente la tendencia es utilizarlo para toda la web.

Para saber más ...

Google lleva desde 2014 impulsando el uso de HTTPS para toda la web. Las páginas web que no ofrecen HTTPS son penalizadas en la lista de resultados del buscador.

Actividad 1.1

Visita la web de un proveedor de hosting (ovh, ionos, arsys...) y observa las opciones de contratación de hosting web y servidores. ¿En qué se diferencian?

2.1.1. URL

Un **URL** (Uniform Resource Locator, Localizador de Recurso Uniforme) es una cadena de texto con la se identifica un recurso de un servidor. En HTTP es la dirección web con la que el navegador "sabe" con qué servidor debe comunicarse y qué recurso (página web, imagen, archivo CSS...) tiene que solicitar. Un URL básica tiene el siguiente formato:

```
protocolo://dominio/recurso
```

- **protocolo**: indica el protocolo a usar. En HTTP es **http** o **https**.
- **dominio**: nombre asociado a la **IP** del servidor. Indica qué servidor aloja la página web. Podemos distinguir dos partes separadas por un punto:
  - **Nombre** de dominio.
  - **TLD** (Top Level Domain, Dominio de Nivel Superior). Los más comunes son *com*, *es*, *net*... pero hay muchos más. Normalmente indican país (.es, .pt, .it, .fr...) o utilidad (.com: comercial, .gov: gubernamental, .edu: educación...).
- **recurso**: indica la ruta y nombre del archivo concreto que se le pide al servidor HTTP. Si no se indica ningún recurso, el servidor puede devolver un archivo por defecto. Normalmente suelen estar configurados para devolver el archivo **index.html**.

Ejemplo: https://www.juntadeandalucia.es/index.html

Para saber más ...

Los siguientes dominios son comunes en internet, pero no son usados para su propósito original:

- *.me: representa al país Montenegro, pero se usa para páginas personales.*
- *.io: representa al Territorio Británico del Océano Índico, pero es muy utilizado para páginas de tecnología, ya que IO es la abreviatura de Input / Output (Entrada / Salida).*
- *.tv: representa a las islas de Tuvalu, pero se usa para registrar sitios relacionados con la Televisión.*
- *.tk: representa a las islas de Tokelau, un territorio neozelandés. Estas islas ofrecen sus dominios gratis para hacerse publicidad, pero no están bien considerados por Google.*

Un URL más complejo tiene el siguiente formato:

```
protocolo://subdominio.dominio:puerto/directorios/
recurso
```

- **subdominio**: etiqueta con la que se puede reutilizar el mismo dominio para varias páginas web.
- **puerto**: Si no se especifica, se utiliza por defecto 80 en HTTP y 443 en HTTPS.
- **directorios**: especifica el subdirectorio de la raíz en el que se encuentra el recurso, si no aparece entonces el recurso está en la raíz del servidor.

Ejemplo: https://www.juntadeandalucia.es:443/temas/estudiar/fp.html

Actividad 1.2

Visita alguna de las webs de la Actividad 1.1 o algún sitio web especializado en dominios como godaddy y comprueba si está disponible algún dominio que se te ocurra. Comprueba varias extensiones (.com, .es, .net...).

2.1.2. Peticiones y respuestas

Cuando introducimos el URL de un documento web en el navegador, se lleva a cabo el siguiente proceso:

1. El navegador envía una petición al servidor solicitando el recurso web.
2. El servidor le responde con el HTML del recurso web
3. El navegador analiza el HTML. Si el HTML incluye referencias a imágenes, hojas de estilo CSS... el navegador hace peticiones al servidor para cada uno de estos archivos.
4. El navegador renderiza (dibuja) el documento web.

Para saber más ...

El navegador no tiene por qué esperar a tener todos los archivos para renderizar la página. Normalmente la página se empieza a dibujar antes de disponer de las imágenes.

El protocolo HTTP es un protocolo sin estado, lo que quiere decir que cada petición que se envía es independiente de la anterior. Además define cómo deben ser esos mensajes con los que el navegador realiza las peticiones y el servidor devuelve las respuestas. Los mensajes son mensajes de texto que constan de:

- Línea inicial: protocolo y
  - Acción (método) si es una petición
  - Código de respuesta si es una respuesta
- Cabeceras del mensaje: metadatos de la petición.
- Cuerpo o contenido del mensaje.

Ejemplo de petición para el URL http://www.example.com/index.html

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:45.0) Gecko/20100101 Firefox/45.0
[Línea en blanco]
```

Ejemplo de respuesta

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 2022 23:59:59 GMT
Content-Type: text/html
Content-Length: 129

<html lang="es">
<head>
<meta charset="utf-8">
<title>Título del sitio</title>
</head>
<body>
<h1>Hola Mundo</h1>
</body>
</html>
```

Métodos de petición

Indican al servidor HTTP la acción a realizar. Hay varios pero nos centraremos en:

- **GET**: solicita una representación de un recurso específico. Las peticiones que usan el método GET sólo deben recuperar datos.
- **POST**: se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor..
- **PUT**: reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
- **DELETE**: borra un recurso en específico.

Códigos de respuesta

El servidor, cuando responde a una petición, incluye en la respuesta un código numérico y una etiqueta que indica el resultado de la petición. Los hay de cinco tipos:

- **1xx**: Respuestas informativas.
- **2xx**: Respuestas correctas.
- **3xx**: Respuestas de redirección.
- **4xx**: Errores causados por el cliente.
- **5xx**: Errores causados por el servidor.

Algunos de los códigos de respuesta más frecuentes son:

- **100 Continue**: Esta respuesta provisional indica que todo está bien hasta este momento y que el cliente debe continuar con la solicitud.
- **102 Processing**: Indica que el servidor está procesando la petición y aún no puede devolver una respuesta.

- **200 OK**: Indica que la petición se ha gestionado correctamente. Es la respuesta más común a una petición si no hay problemas.
- **301 Moved Permanently**: Indica que el recurso solicitado ha cambiado de URL. La respuesta suele incluir una cabecera con la nueva URL del recurso.
- **304 Not Modified**: Indica que el recurso no ha cambiado desde la última vez que se solicitó.
- **307 Temporary redirect**: Similar a 301, sólo que el cambio de URL es temporal.
- **400 Bad Request**: Indica que la petición está mal formada y el servidor no la puede procesar.
- **403 Forbidden**: Indica que no tiene permiso para acceder al recurso solicitado.
- **404 Not Found**: Indica que el recurso solicitado no existe en el servidor. Es uno de los códigos de error más reconocibles.
- **500 Internal Server Error**: Indica que el servidor ha tenido un error interno al procesar la petición.
- **503 Service Unavailable**: Indica que el servidor no está listo para atender la petición. Es el mensaje que suele devolver el servidor cuando está sobrecargado.

Para saber más ...

*Uno de los errores más divertidos es el **418: I'm a teapot** (soy una tetera), que es devuelto cuando a una tetera se le ordena preparar café. Este error es un guiño al Hyper Text Coffee Pot Control Protocol, protocolo que se creó en 1998 como broma del April Fools (equivalente al día de los inocentes en España).*

Cabeceras

Aportan información adicional sobre la petición o la respuesta en sí. Algunas de las cabeceras más frecuentes son:

- Sobre cómo debe responder el servidor al cliente:
  - **Accept**: Tipo de MIME (tipo de dato) aceptado por el cliente.
  - **Accept-Charset**: código de caracteres aceptado por el cliente.
  - **Accept-Encoding**: método de compresión aceptado por el cliente.
  - **Accept-Language**: indica el idioma aceptado por el cliente.
- Sobre el contenido:
  - **Content-Type**: indica el MIME del contenido.
  - **Content-Length**: longitud en caracteres del mensaje.
  - **Content-Encoding**: indica la codificación de caracteres del mensaje.
  - **Content-Language**: indica el idioma del mensaje.
- Sobre fechas:
  - **Date**: fecha de creación del mensaje.
  - **Last-Modified**: fecha en la que se modificó por última vez el recurso.

- Otros
    - **User-Agent**: Describe al cliente (sistema operativo, navegador...)
    - **Host**: indica servidor destino del mensaje.
    - **Location**: indica el URL donde se encuentra el recurso.

Para saber más ...

El tipo MIME (Multipurpose Internet Mail Extensions, Extensiones Multipropósito de Correo de internet) es una etiqueta que define el tipo de dato que se está enviando. Ejemplos de MIME son text/html, image/jpeg, video/mp4...At Home.
En la documentación de mozilla puedes consultar un listado más completo.

2.2. Programación en cliente y servidor

A la hora de desarrollar código para una aplicación web, ese código puede ser ejecutado en dos entornos:

- **Entorno servidor** (también conocido como **Backend**): El código **se ejecuta en el servidor** y el resultado de la ejecución es enviado al cliente. Este tipo de desarrollo **aumenta la carga de trabajo del servidor**, pero **mantiene el código bajo control del servidor**, oculto al cliente. Ejemplo: PHP.
- **Entorno cliente** (también conocido como **Frontend**): El servidor envía el código al cliente y **se ejecuta en el navegador**. Este tipo de desarrollo **aligera la carga de trabajo del servidor** (cada cliente ejecuta el código en su navegador) pero **pone el código bajo control del cliente**, el cual puede verlo e incluso modificarlo. Ejemplo: Javascript.

*En el ámbito de las aplicaciones web como por ejemplo una tienda online o un blog, también se denomina **Frontend** a la parte pública de la web y **Backend** a la zona reservada para administradores.*

Para saber más ...

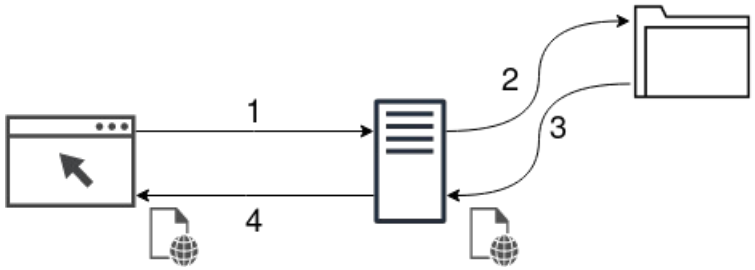
Durante el auge de las criptomonedas, un ataque muy común a los servidores web consistía en modificar sus páginas web para que incluyeran un código JavaScript que minara Bitcoins, de forma que el visitante, sin saberlo, ejecutara en su dispositivo dicho código mientras visitaba la página.
Los usuarios más celosos de la seguridad y la privacidad recomiendan bloquear por defecto el código JavaScript en el navegador.

Actividad 1.3

Los navegadores actuales integran herramientas de desarrolladores con las que puedes, entre otras cosas, editar el código JavaScript de las páginas y controlar su ejecución. Utilízalas para visualizar el código JavaScript utilizado en alguna página web.

3. Generación dinámica de páginas Web. Ventajas

Las páginas web más básicas y tradicionales consisten en archivos de texto que contienen el código HTML de la página web. Este tipo de páginas web se denominan **páginas web estáticas**. En este tipo de páginas, la secuencia que se sigue es la siguiente:

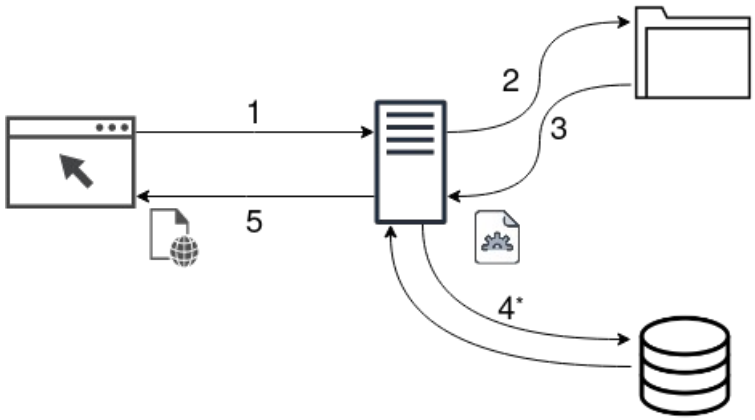


1. El navegador solicita una página web al servidor.
2. El servidor busca el recurso solicitado en sus carpetas.
3. El servidor encuentra el archivo HTML solicitado.
4. El servidor ofrece el archivo HTML al navegador.

Se caracterizan por:

- Son fáciles de desarrollar ya que no hay que saber programar. Sólo se necesita saber HTML y CSS, aunque hay software de diseño web que nos permite crear páginas sin necesidad de utilizar dichos lenguajes.
- Si se dispone del código HTML se puede visualizar directamente con el navegador, no se necesita un servidor para poder verlas.
- Su contenido no varía. Si se desea actualizar el contenido de la página, hay que hacerlo manualmente.

Estas páginas por tanto son sencillas, pero no son adecuadas si la página se debe actualizar con frecuencia. Para este tipo de sitios web surgieron las **páginas web dinámicas**. Son páginas cuyo contenido HTML es generado una aplicación que ejecuta el servidor cada vez que el cliente realiza la petición. En este tipo de páginas, la secuencia que se sigue es la siguiente:



1. El navegador solicita una página web al servidor.
2. El servidor busca el recurso solicitado en sus carpetas.
3. El servidor encuentra el archivo de código solicitado y lo ejecuta.
4. *Opcionalmente, durante la ejecución del código se hacen peticiones a bases de datos.
5. El servidor ofrece el archivo HTML generado por el código al navegador.

Se caracterizan por:

- Son complejas de desarrollar ya que necesitan conocimientos de programación.
- Se necesita un servidor que ejecute el código servidor que genera la página para poder obtener el código HTML.
- Se generan cada vez que el cliente hace una petición, de forma que siempre están actualizadas.
- Frecuentemente hacen uso de bases de datos para generar páginas web a partir de la información almacenada.
- Un mismo código puede generar páginas web distintas, dependiendo de factores como la fecha y hora, el visitante o la información (producto, noticia...) que se quiera consultar.

Actividad 1.4

Enumera los pasos que se siguen si deseo acceder a la página dinámica `http://www.miblog.com/estadisticas.php`*, que utiliza información almacenada en una base de datos. ¿Y si esa página incluye una imagen cuyo URL es* `http://www.miblog.com/logo.png`*?*

4. Lenguajes de programación en entorno servidor

Una de las elecciones fundamentales a la hora de desarrollar una aplicación es qué lenguaje de programación elegir. Para ello nos podemos fijar en qué tipos hay y cuáles son sus características:

- Lenguajes según su finalidad:
    - **Específicos para desarrollo web**: Son lenguajes diseñados específicamente para desarrollar aplicaciones web, por lo que disponen de muchos recursos (funciones, librerías, documentación...) que nos facilitan el desarrollo. Ejemplo: PHP.
    - **De propósito genera**l: sirven para desarrollar cualquier tipo de aplicación o no están especializados en algún campo concreto. En este caso, normalmente no se utilizan por sí solos para desarrollo web, sino que se utilizan junto a un Framework, que es una plataforma que nos proporciona gran parte de la estructura, librerías, recursos... sobre la que podemos desarrollar la aplicación. Ejemplo: Java junto al framework Spring.

- Lenguajes según su ejecución:
    - **Compilados**: necesitan una aplicación (el compilador) que a partir del código fuente genere un archivo binario que el ordenador pueda ejecutar directamente. Son rápidos en ejecutar pero sólo se pueden ejecutar en la arquitectura para la que han sido compilados. Ejemplo: Go.
    - **Interpretados**: necesitan una aplicación (el intérprete) que vaya leyendo y ejecutando el código línea a línea. Son más lentos de ejecutar, pero se pueden ejecutar en cualquier arquitectura que disponga de un intérprete. Ejemplo: PHP, JavaScript, Ruby.
    - **Compilados a código intermedio**: es una mezcla de los dos estilos anteriores. El código es compilado a un código intermedio, independiente de la plataforma, y este es ejecutado por un intérprete o una máquina virtual. De esta manera obtenemos un código más rápido que el interpretado pero más portable que el compilado. Ejemplo: Java (ejecutado en la Máquina Virtual de Java, o JVM), C# (ejecutado en la máquina virtual .NET).

Sin embargo no se puede decir que un lenguaje sea el mejor, ya que cada uno tiene sus fortalezas y sus debilidades, y el rendimiento en general suele ser bueno. Por ello, la elección de un lenguaje de programación en la práctica viene determinado más bien por otros factores como:

- Familiaridad con el lenguaje: los desarrolladores suelen preferir utilizar un lenguaje que ya dominan. Es algo a sopesar ya que, con un lenguaje nuevo, el programador necesita tiempo para aprenderlo y no tiene tanta seguridad en el desarrollo al no tener experiencia.

- Comunidad y desarrollos existentes: que haya una gran comunidad de desarrolladores de un lenguaje en concreto supone que encontraremos mucha documentación, librerías... que podremos utilizar o consultar.
- Alguna característica determinante: hay proyectos en los que un factor como la escalabilidad o el rendimiento es crucial, y para ello se suelen escoger lenguajes que destaquen en ese aspecto.
- Imposición: en ocasiones el proyecto exige usar una herramienta o librería determinada y, por tanto, nos vemos en la obligación de usar un lenguaje compatible.

Los principales lenguajes y tecnologías utilizadas en desarrollo web backend son:

- **PHP**: (PHP: Hypertext Preprocessor) es un lenguaje de programación especializado en el desarrollo de páginas web. Se puede utilizar sólo o con frameworks como Laravel, Symfony, CodeIgniter, o CakePHP. Gran cantidad de CMS como WordPress, Joomla, Drupal, Magento o Prestashop funcionan con PHP. Sus puntos débiles eran la velocidad y la escalabilidad, sin embargo con la versión PHP 7 se han mejorado estos aspectos.
- **JavaScript y Node.js**: JavaScript es un lenguaje diseñado para ser ejecutado en el navegador, de hecho es el más popular de los lenguajes de desarrollo en entorno cliente. Sin embargo Node.js es un framework para desarrollo en entorno servidor que funciona con JavaScript y es muy valorado por su escalabilidad.
- **Java y Spring Boot**: Por sí solo, es un lenguaje de propósito general. Junto con el framework Spring Boot, permite desarrollar aplicaciones web.
- **C# y ASP.NET**: ASP.NET es un framework para desarrollar páginas web que permite utilizar lenguajes de Microsoft como Visual .NET o C#.
- **Python y Django**: Python es un lenguaje de propósito general, muy valorado por su sencillez. Junto con el framework Django, permite la elaboración de aplicaciones web.
- **Ruby y Ruby on Rails**: Ruby es un lenguaje de programación de propósito general inspirado en Perl. Junto con el framework **Ruby on Rails** (conocido también como RoR o Rails) permite desarrollar aplicaciones web.

Actividad 1.5

Elige uno de los lenguajes o frameworks reseñados en este punto e investiga en internet su popularidad y cuáles son sus fortalezas y sus ventajas comparados con otras alternativas. Utiliza Google Trends (https://trends.google.es/trends/?geo=ES) para comparar la popularidad. Pon en común la información encontrada con el resto de la clase.

5. Integración con los lenguajes de marcas

Las páginas web dinámicas se crean a partir de una aplicación que genera el código HTML correspondiente. En un lenguaje de programación de propósito general deberemos hacerlo utilizando comandos de impresión. En el siguiente ejemplo, una aplicación hecho en python imprime una web con el resultado de sumar dos variables:

```
#!/usr/bin/env python
num1 = 2
num2 = 3
print("<html>")
print("<head>")
print("<title>Ejemplo</title>")
print("</head>")
print("<body>")
print("<h1>Hola mundo</h1>")
print("<p>" + str(num1) + " + " + str(num2) + " = " + str(num1 + num2) + "</p>")
print("</body>")
print("<html>")
```

Este es el HTML generado por la aplicación:

```
<html>
<head>
<title>Ejemplo</title>
</head>
<body>
<h1>Hola mundo</h1>
<p>2 + 3 = 5</p>
</body>
<html>
```

Para generar un documento HTML o de cualquier otro lenguaje de marcas, el código estará lleno de sentencias "print", "echo", "write" o similares, que entorpecerá el desarrollo web.

Sin embargo, algunos lenguajes especializados en desarrollo web permiten la integración con los lenguaje de marcas, de forma que el código se incrusta en el lenguaje de marcas, únicamente en la parte en la que se necesite. El siguiente ejemplo implementa la mismo aplicación que el ejemplo anterior utilizando PHP:

```
<?php
$num1 = 2;
$num2 = 3;
?>
<html>
<head>
<title>Ejemplo</title>
</head>
<body>
<h1>Hola mundo</h1>
<p><?php print($num1 . " + " . $num2 . " = " . $num1+$num2;?></p>
</body>
<html>
```

La integración con los lenguajes de marcas permite escribir el código únicamente donde se necesita, mejorando la legibilidad de este.

6. Integración con los servidores Web

Como hemos visto en apartados anteriores, un servidor web es una aplicación que recibe y contesta peticiones HTTP. Aunque las principales opciones son **Apache** y **Microsoft IIS**, existe variedad de ellos, diferenciándose en aspectos como la licencia, los lenguajes soportados o la ligereza. Ejemplos de software servidor HTTP son:

- **Apache**: Apache HTTP Server (conocido como Apache) es el servidor HTTP más usado, aunque en los últimos años ha perdido cuota de mercado. Es código abierto y extensible. Frecuentemente se usa con MySQL y PHP, lo que se conoce como plataforma AMP (Apache, MySQL y PHP) y a la que le suele anteponer la letra del sistema operativo en el que se monta, por ejemplo **LAMP** en el caso de Linux o WAMP en el caso de Windows. Soporta multitud de lenguajes gracias a sus módulos.

Lenguaje	Módulo que lo habilita en Apache
PHP	mod_php
Python	mod_wsgi
Perl	mod_perl
Ruby	mod_passenger

- **Nginx**: servidor conocido por su velocidad y alto rendimiento, aunque no soporta algunas características de Apache. Soporta multitud de lenguajes utilizando módulos instalables.
- **Microsoft IIS**: servidor especializado en tecnología de Microsoft, aunque es compatible con otros lenguajes. Se incluye con Windows Server.
- **Lighttpd**: servidor web de bajo consumo y rápido.

Algunos frameworks incluyen su propio servidor web, de modo que no necesitan instalar ninguno.

7. Herramientas de desarrollo

7.1 IDE

Para programar, en principio, sólo necesitamos un editor de texto con el que escribir el código. Sin embargo, para desarrollar una aplicación es muy recomendable utilizar un IDE (Integrated Development Environment, Entorno de Desarrollo Integrado), una aplicación que, además del editor de código, incluye otras herramientas necesarias o muy útiles para el desarrollo. Entre estas características se encuentran:

- **Resaltado de texto**. Muestra con distinto color o tipo de letra los diferentes elementos del lenguaje: sentencias, variables, comentarios, etc. También genera indentado automático para diferenciar de forma clara los distintos bloques de una aplicación.
- **Completado automático**. Detecta qué estás escribiendo y cuando es posible te muestra distintas opciones para completar el texto.
- **Navegación en el código**. Permite buscar de forma sencilla elementos dentro del texto, por ejemplo, definiciones de variables.
- **Comprobación de errores al editar**. Reconoce la sintaxis del lenguaje y revisa el código en busca de errores mientras lo escribes.
- **Sugerencias para mejora del código**: El IDE detecta variables que no se usan, líneas que no cumplan un estilo de codificación...
- **Generación automática de código**. Ciertas estructuras, como la que se utiliza para las clases, se repiten varias veces en una aplicación. La generación automática de código puede encargarse de crear la estructura básica, para que sólo tengas que rellenarla. Estos fragmentos de código se conocen como **snippets**.
- **Ejecución, depuración y análisis del rendimiento**. Esta característica es una de las más útiles. El IDE se puede encargar de ejecutar una aplicación para poder probar su funcionamiento. Cuando algo no funciona, te permite depurarlo (**debug**) con herramientas como la ejecución paso a paso, el establecimiento de puntos de ruptura o la inspección del valor que almacenan las variables. Además, permite medir el rendimiento (**profiling**) para optimizar la eficiencia del código.
- **Gestión de versiones**. En conjunción con un sistema de control de versiones, el entorno de desarrollo te puede ayudar a guardar copias del estado del proyecto a lo largo del tiempo, para que si es necesario puedas revertir los cambios realizados.

7.2. Herramientas de peticiones HTTP

Para comprobar si nuestra aplicación web funciona, lo más común es utilizar el navegador para acceder a las páginas web que la componen. Sin embargo, habrá ocasiones en las que necesitaremos controlar el método de la petición (GET, POST, DELETE...), los parámetros de la petición...

Para estos casos existen extensiones que nos permiten realizar peticiones totalmente configurables y observar los resultados. Postman es un ejemplo de ello.

7.3. Herramientas de test

Las pruebas, los tests... han sido una fase del desarrollo que tradicionalmente se dejaba para el final, llegando incluso a omitirse si el lanzamiento de la aplicación era urgente. Sin embargo, las corrientes actuales de desarrollo proponen darle a las pruebas un papel protagonista, llegando incluso a guiar el desarrollo del proyecto (metodología TDD), ya que de esta manera el código es más seguro y mantenible.

Existen aplicaciones, frameworks... con los que realizar pruebas automatizadas a lo largo del desarrollo. Ejemplos de estas herramientas son Selenium, TestCafe o Cucumber.

7.4. Docker

Para desarrollar una aplicación web lo más común es contar con un servidor local en el comprobar lo que se va desarrollando, y si se trata de un equipo de desarrolladores es muy probable que cada uno tenga su propio servidor local. En cada uno de estos servidores habrá que instalar y configurar multitud de software: servidor, bases de datos, lenguaje de programación... y esto puede suponer un problema, ya que como cada servidor es distinto (pueden incluso no tener el mismo sistema operativo), puede que no tengan exactamente la misma versión del software instalado. Esto puede parecer un problema menor, pero da lugar a situaciones en las que un mismo código funciona bien en un servidor y en otro no o, peor aún, que el código funcione en los servidores locales pero luego falle en el servidor de producción.

Una forma de resolver este problema es mediante la creación de ***máquinas virtuales***, de forma que todos los miembros del equipo de desarrollo utilicen la misma máquina virtual como servidor local. Sin embargo hay una solución más ligera que lleva unos años ganando popularidad: ***Docker***.

Docker es un software de despliegue de aplicaciones mediante contenedores. Estos contenedores son entornos virtuales en los que se instala el software, con su propia configuración y librerías necesarias. Es decir, en lugar de virtualizar un ordenador completo, se virtualiza únicamente la aplicación.

De esta manera se solucionan problemas derivados de trabajar con versiones distintas, se unifica el entorno de desarrollo y permite automatizar pruebas en distintos entornos.