

UD1. Selección de arquitecturas y herramientas de programación

1. Desarrollo Web

La web (www - World Wide Web) nació como tal en 1989 (Berners-Lee - CERN). En una década su crecimiento fue exponencial, por ello el consorcio W3C (World Wide Web Consortium) comenzó a desarrollar estándares que asegurasen el desarrollo de la web a largo plazo. Aunque en 2019, el Consorcio WWW declinó a favor de WHATWG (Web Hypertext Application Technology Working Group) en cuanto al control de las futuras normas acerca de HTML y DOM.

La web no es más que una representación de un universo de información accesible a través de internet, haciendo uso de una serie de recursos interconectados:

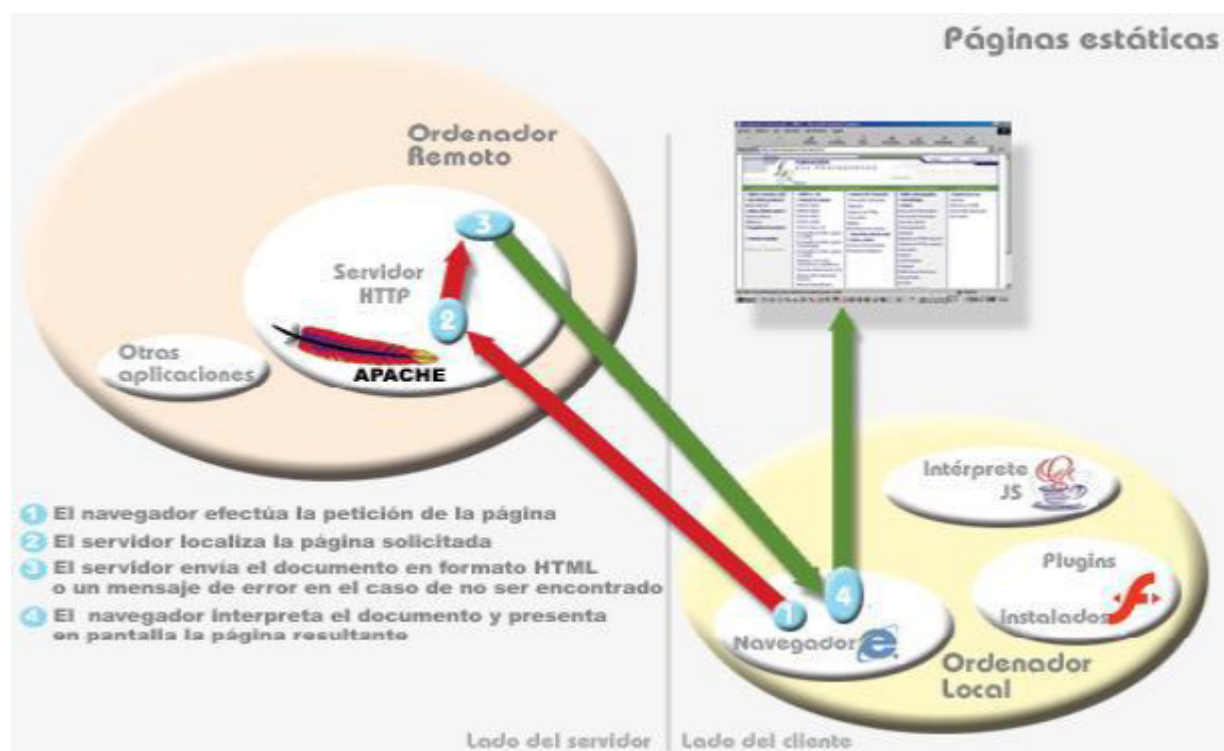
- Componentes físicos: hubs, repetidores, puentes, routers, ...
- Protocolos de comunicación: TCP, IP, HTTP, FTP, ...
- Sistema de nombres de dominio (DNS)
- Software para proveer y consumir dichos recursos: servidores y clientes.

Es impensable en la actualidad un mundo sin internet, ya que la mayoría de la actividad comercial, profesional o lúdica se sustentan en la web, de ahí la importancia y alta demanda de profesionales del sector de desarrollo web.

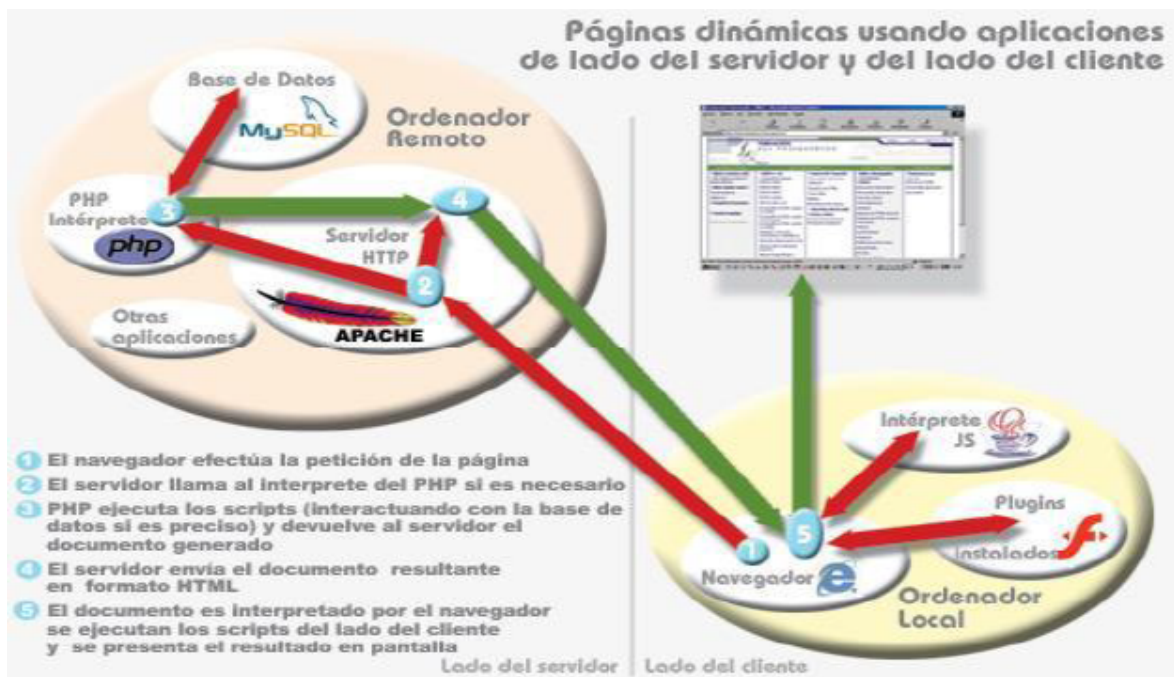
La configuración arquitectónica habitual suele ser Cliente/Servidor:

- CLIENTE: Componente consumidor de servicios.
- SERVIDOR: Proceso proveedor del servicio.

Para páginas web estáticas se tendrá:



Para páginas web dinámicas o **aplicaciones web** se tendrá:



Es por esta arquitectura cliente/servidor que, aunque el desarrollo web es muy diverso, los trabajos de este sector se suelen dividir en dos aspectos: desarrollo web del lado del cliente (front-end) y desarrollo web del lado del servidor (back-end).

1.1. Desarrollo web del lado del cliente y del lado del servidor.

Se conoce como **front-end** a la parte de un sitio web que interactúa con los usuarios, de ahí que se denomine desarrollo del lado del cliente. Frontend es la parte de una aplicación a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el **navegador** y que se encargan de la interactividad con los usuarios.

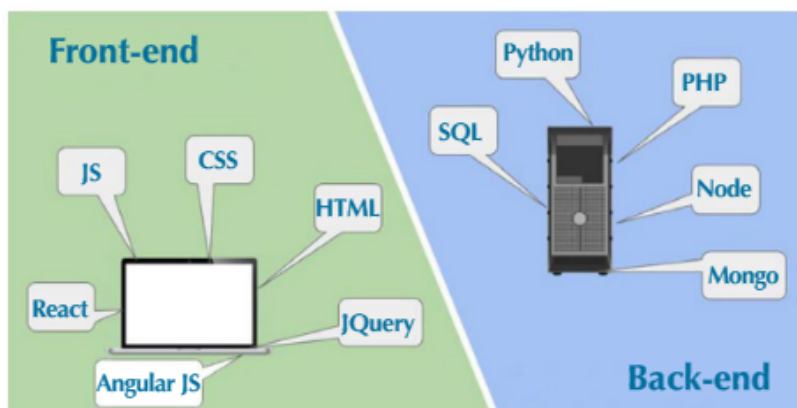
Para convertirte en *Frontend Developer* debes saber HTML y CSS, los lenguajes de maquetación que nos permiten definir la estructura y estilos de una página web. Y también JavaScript, un lenguaje de programación para definir la lógica de nuestra aplicación, recibir las solicitudes de los usuarios y enviárselos al backend.

Una vez dominadas estas tecnologías se puede iniciar el uso de algunos frameworks, librerías o preprocesadores que expanden las capacidades para crear todo tipo de interfaces de usuario y dotarlos de funcionalidad e interactividad. Algunos de ellos son

- Para JavaScript: React, Vue, Angular, Svelte...
- Para CSS: Bootstrap, Tailwinds, Foundation, Sass, Less, Stylus, PostCSS...

Backend es la parte del sitio web que trata con el **servidor** y base de datos, se trata por tanto de la capa de acceso a datos de una aplicación web, que no es directamente accesible por los usuarios, además contiene la lógica de la aplicación que maneja dichos datos. El backend tiene acceso al servidor, de manera que entiende la forma en que el navegador realiza las peticiones.

Algunos de los lenguajes de programación para back-end son PHP, Python, Go, Ruby...Y así como en el front-end, todos estos lenguajes tienen diferentes frameworks que permiten trabajar mejor según el proyecto que se está desarrollando, como Laravel, Yii, Django, Flask, Symphony, Ruby on Rails...



2. Navegadores Web

Como acabamos de ver, el desarrollo web en entorno cliente se fundamenta en tecnologías que corren en los navegadores web, por lo tanto, es fundamental conocer las principales características de estos.

2.1. Definición y características

Un navegador web es un componente software o aplicación distribuida, habitualmente como software libre, que permite a un usuario acceder (y, normalmente, visualizar) a un recurso publicado por un servidor Web a través de internet y descrito mediante una dirección URL (Universal Resource Locator).

Por tanto, la función principal de un navegador es solicitar al servidor los recursos web que elija el usuario y mostrarlos en una ventana.

El recurso suele ser un documento HTML, pero también puede ser un archivo PDF, una imagen o un objeto de otro tipo. El usuario especifica la ubicación del recurso mediante el uso de una URI (siglas de Uniform Resource Identifier, identificador uniforme de recurso).

La forma en la que el navegador interpreta y muestra los archivos HTML se determina en las especificaciones de CSS y HTML establecidas por el consorcio W3C y posteriormente WHATWG.

La evolución de los navegadores desde los años 90 hasta ahora ha sido enorme, desde simples visualizadores de texto a los actuales navegadores repletos de funcionalidades.

Mosaic

- Uno de los primeros navegadores y el primero con capacidades gráficas.
- Inicialmente ejecutado sobre UNIX, posteriormente al resto de las plataformas.
- Base para las primeras versiones de Internet Explorer y Mozilla.

Netscape Navigator (Communicator)

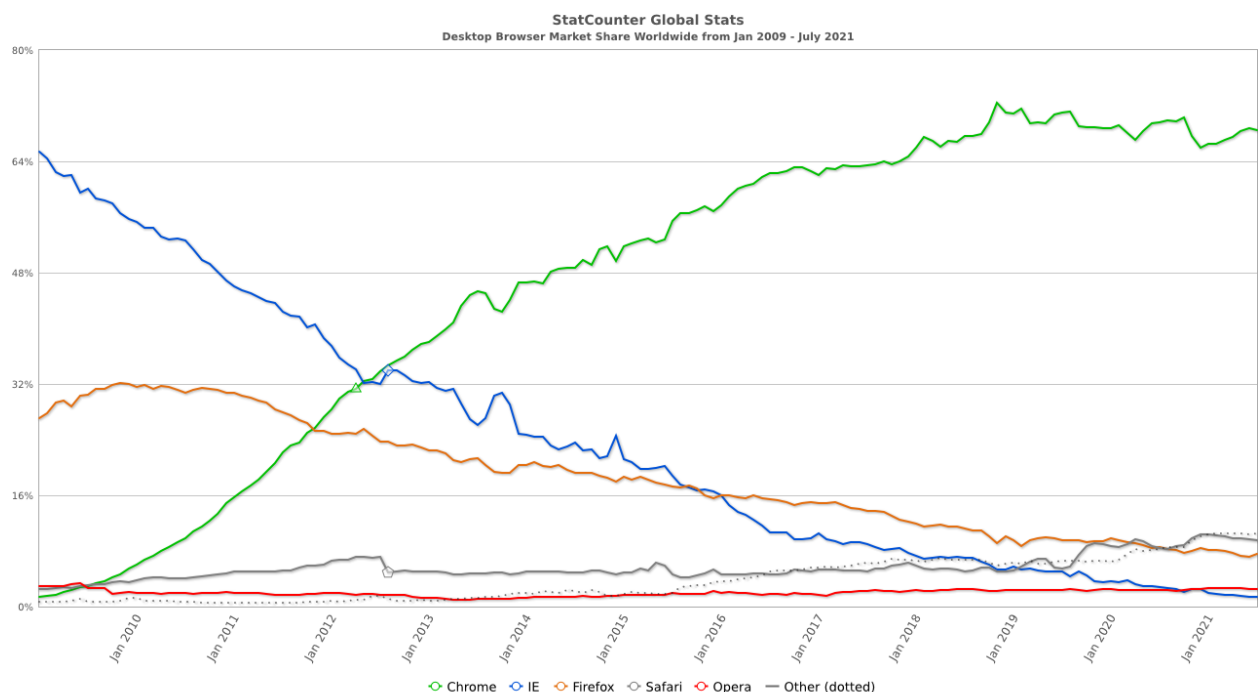
- Primero en incluir un módulo para la ejecución de código script (JavaScript).
- “Perdedor” en la “guerra de los navegadores” frente a Microsoft (dominio de éste a finales de los 90).
- Características base para Mozilla Firefox.

Internet Explorer

- Navegador de Microsoft.
- Tuvo una cuota de distribución y uso elevada gracias a su integración en Windows.
- Descenso frente Firefox o Chrome.
- Última versión IE11 (finales 2013), con soporte para estándares web, personalización de navegación y seguridad. Reemplazado por Microsoft Edge

En la actualidad los más populares (escritorio), según la web statcounter, son:

- Google Chrome (68%)
- Safari (9,5%)
- Edge (8,5%)
- Mozilla (7,5%)
- Opera (2,5%)



Fuente: <https://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-200901-202107>

Google Chrome

- Creado en 2008, es el navegador de Google compilado a partir de componentes de código abierto. Basado en [Chromium](#).
- Seguridad, velocidad y estabilidad gracias a que sigue una arquitectura multiproceso en la que cada pestaña se ejecuta de forma independiente.
- Motor de renderizado: Blink
- Intérprete JS: V8

Safari

- Navegador por defecto del sistema Apple.
- Las últimas versiones incorporan la navegación por pestañas, corrector ortográfico en formularios, almacenamiento de direcciones favoritas (“marcadores”), bloqueador de ventanas emergentes, soporte para motores de búsqueda personalizado o un gestor de descargas propio.
- Motor de renderizado: WebKit
- Intérprete JS: JavaScriptCore

Microsoft Edge

- Desarrollado por Microsoft, basado en [Chromium](#).
- Fue lanzado por primera vez para Windows 10 y Xbox One en julio de 2015.
- Motor de renderizado: Blink
- Intérprete JS: V8

Mozilla Firefox

- De código abierto, multiplataforma, de gran aceptación en la comunidad de desarrolladores web.
- Gran variedad de utilidades, extensiones y herramientas para la personalización y apariencia del navegador.
- Fue de los primeros en incluir la navegación por pestañas.
- Motor de renderizado: Gecko
- Intérprete JS: SpiderMonkey

Opera

- La primera versión pública del software vio la luz el 30 de abril de 1996 la cual solo funcionaba en Microsoft Windows. Basado en [Chromium](#).
- Motor de renderizado: Blink
- Intérprete JS: V8

Comparativa de consumo de recursos de navegadores: <https://www.youtube.com/watch?v=1j9uRMipdAE>

2.2. Arquitectura del navegador:

Un navegador puede ejecutarse como un proceso con muchos hilos/subprocesos diferentes o como muchos procesos diferentes con algunos hilos/subprocesos que se comunican entre sí:

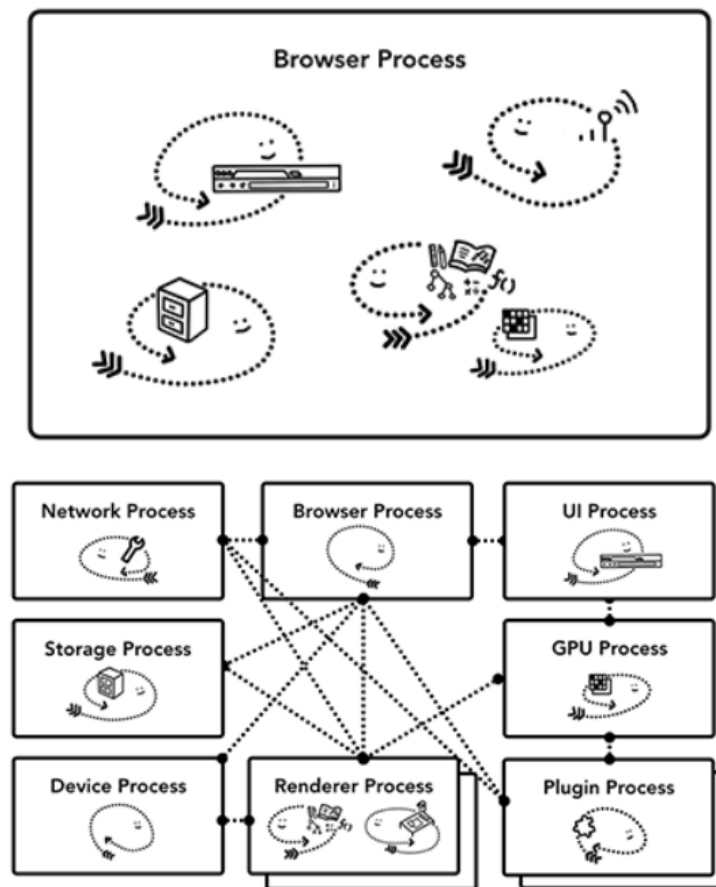


Figura 1. Diferentes arquitecturas de navegador en diagrama proceso/hilo (developers.google.com)

No hay una norma establecida sobre cuál debe ser la arquitectura. Si nos centramos en Chrome, su arquitectura se puede representar:

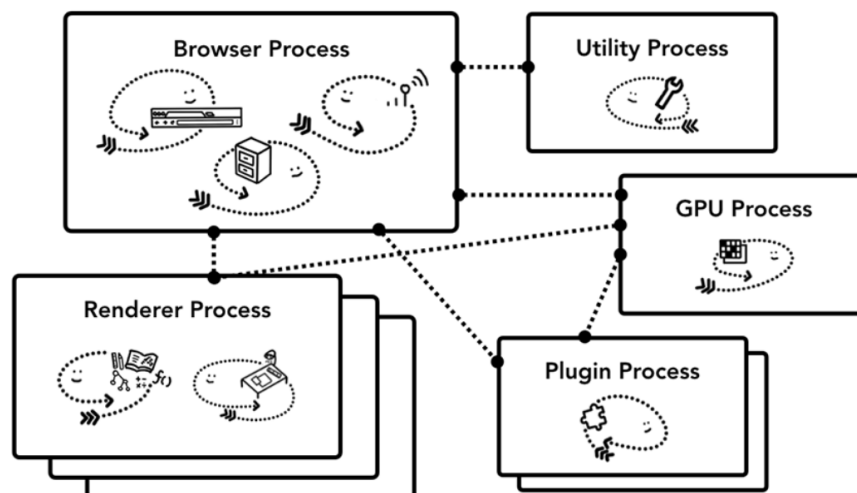


Figura 2. Diagrama de la arquitectura multiproceso de Chrome.

El proceso de renderizado de Chrome se muestra en la figura con varias capas, representando a Chrome ejecutando varios procesos de renderizado, uno por cada pestaña.



Procesos principales

Navegador	Controla la parte propia del navegador de la aplicación, incluida la barra de direcciones, los marcadores, los botones de avance y retroceso. También maneja las partes invisibles y privilegiadas de un navegador web, como las solicitudes de red y el acceso a archivos.
Renderizador	Controla cualquier cosa que se visualiza dentro de la pestaña donde se muestra un sitio web. Clásicamente los motores de renderizado predominantes han sido WebKit (Chrome, Safari) y Gecko (Firefox), aunque en la actualidad es Blink (bifurcación de WebKit) el que predomina (Chrome, Opera, Edge..).
Plugin	Controla los complementos utilizados por el sitio web, por ejemplo, flash.
GPU	Maneja las tareas de la GPU de forma aislada de otros procesos. Se divide en diferentes procesos porque las GPU manejan solicitudes de múltiples aplicaciones y las dibujan en la misma superficie.

Para ver cuántos procesos se están ejecutando en Chrome, hacemos clic en el icono del menú de opciones en la esquina superior derecha, seleccionamos “Más herramientas”, luego “Administrador de tareas”. Esto abre una ventana con una lista de procesos que se están ejecutando actualmente y cuánta CPU/memoria están usando.

FLUJO DE NAVEGACIÓN

Veamos un caso de uso simple de la navegación web:

- Escribir una URL en un navegador.
- El navegador obtiene datos de Internet y muestra una página.

Todo comienza con un proceso navegador, que tiene subprocesos como:

- **Subproceso de interfaz de usuario** que dibuja los botones o campos de entrada del navegador (no del interior de la pestaña), es decir la barra de herramientas y el proceso de carga..
- **Subproceso de red:** Se ocupa de la pila de red para recibir datos de internet. Implementa los protocolos de transferencia de ficheros y documentos (HTTP, FTP, ...). Identifica la codificación de los datos obtenidos en función de su tipo (texto, audio, vídeo, ...) codificado en estándar MIME (Multipurpose Internet Mail Extensions).
- **Subproceso de almacenamiento:** Controla el acceso a archivos. y almacén de diferentes tipos de datos para los principales subsistemas del navegador. Suelen estar relacionados con el almacenamiento de historiales de navegación y mantenimiento de sesiones de usuario en disco. Incluye las preferencias de configuración del navegador o la lista de marcadores. A bajo nivel, este sistema administra también los certificados de seguridad y las cookies.

Veamos el proceso paso a paso:

- **Paso 1. Iniciando la navegación:** Cuando el usuario escribe una URL en la barra de direcciones, su entrada es manejada por el hilo de la interfaz de usuario del proceso del navegador. Cuando el usuario presiona Enter, este hilo o subproceso inicia una llamada de red para obtener el contenido del sitio

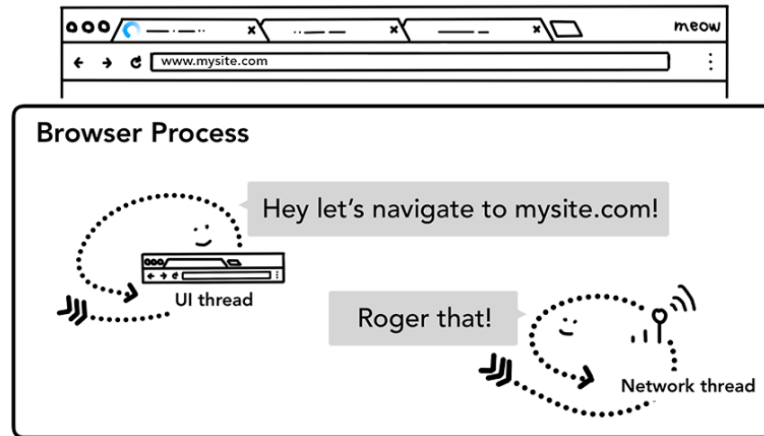


Figura 2. El subproceso UI hablando con el subproceso de red para navegar a mysite.com

- **Paso 2. Leer la respuesta:** Si todo va bien y se produce una respuesta por parte del sitio, una vez que el cuerpo de respuesta (carga útil) comienza a aparecer, el hilo de red mira los primeros bytes del flujo. El encabezado Content-Type de la respuesta debe indicar qué tipo de datos son.

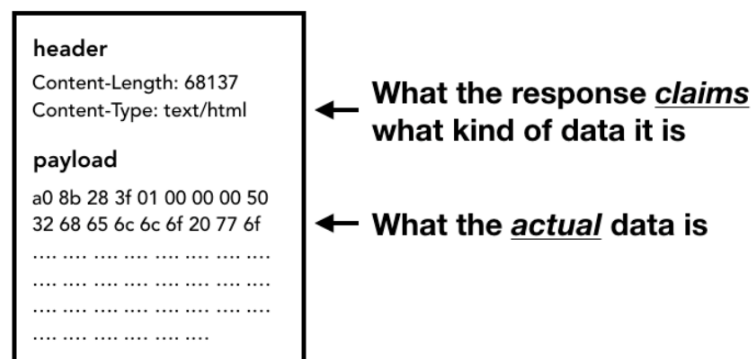


Figura 3 . Encabezado de respuesta que contiene el Content-Type y el payload que son los datos reales.

Si la respuesta es un archivo HTML, entonces el siguiente paso sería pasar los datos al proceso del renderizador, pero si es un archivo zip o algún otro archivo, eso significa que es una solicitud de descarga, por lo que deben pasar los datos a gestor de descargas.

Aquí también es donde ocurre la verificación de [SafeBrowsing](#). Si el dominio y los datos de respuesta parecen coincidir con un sitio malicioso conocido, el hilo de la red alerta para mostrar una página de advertencia. Además, la [verificación de bloqueo de Cross Origin Read Blocking \(CORB\)](#) se realiza para asegurarse de que los datos confidenciales entre sitios no lleguen al proceso de renderizado.

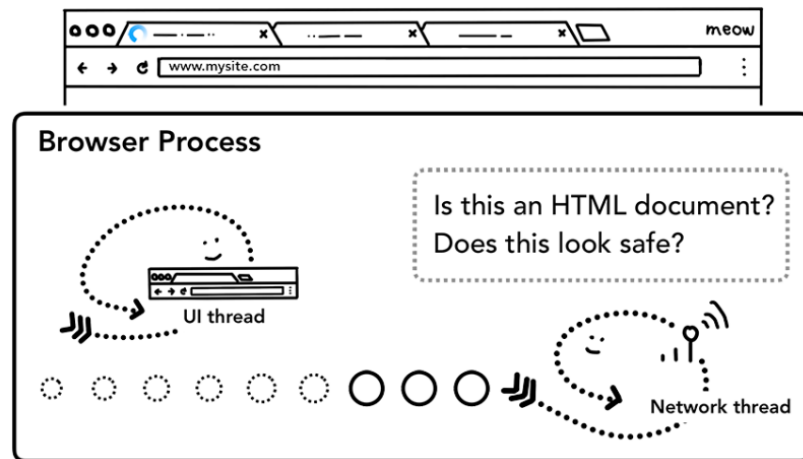


Figura 4. Hilo de red que pregunta si los datos de respuesta son HTML de un sitio seguro.

- Paso 3. Encontrar un proceso de renderizado:** Una vez que se realizan todas las comprobaciones y el hilo de red confía en que el navegador debe navegar al sitio solicitado, el hilo de red le dice al hilo de la interfaz de usuario que los datos están listos. Entonces, el hilo de la interfaz de usuario debe encontrar un proceso de renderizado para continuar con la renderización de la página web.

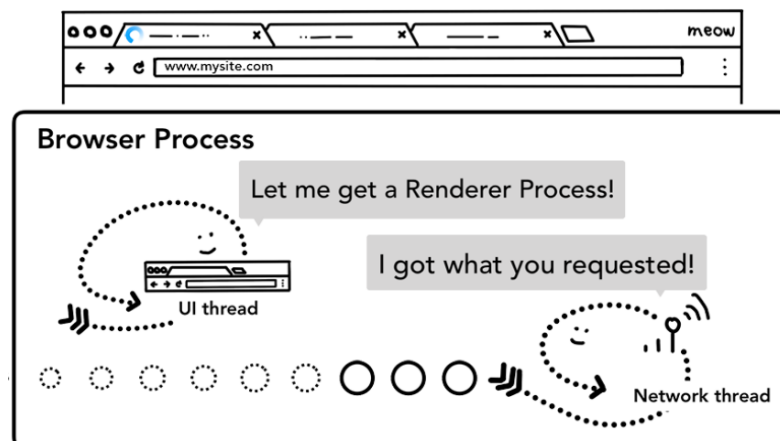


Figura 4. Subproceso de red que indica al subproceso UI que busque el proceso de renderizado.

Dado que la solicitud de red puede tardar varios cientos de milisegundos en obtener una respuesta, se aplica una optimización para acelerar este proceso. Cuando el subproceso de la interfaz de usuario envía una solicitud de URL al subproceso de la red en el paso 1 ya sabe a qué sitio están navegando. El subproceso de la interfaz de usuario intenta buscar o iniciar de forma proactiva un proceso de renderizado en paralelo a la solicitud de red. De esta manera, si todo sale como se esperaba, un proceso de renderizado ya estará en la posición de espera cuando el hilo de la red recibe los datos.

- Paso 4. Confirmar la navegación:** Ahora que los datos y el proceso del renderizador están listos, se envía un IPC (Inter Process Communication) desde el proceso del navegador al proceso del renderizador para confirmar la navegación. También transmite el flujo de datos para que el proceso del renderizador pueda seguir recibiendo datos HTML. Una vez que el proceso del navegador escucha la confirmación del proceso de renderizado, la navegación se completa y comienza la fase de carga del documento.

En este punto, la barra de direcciones se actualiza y el indicador de seguridad y la interfaz de usuario de la configuración del sitio reflejan la información del sitio de la nueva página. El historial de sesiones de la pestaña se actualizará, por lo que los botones de retroceso/avance recorrerán el sitio al que se acaba de navegar. Para facilitar la restauración de la pestaña/sesión cuando cierra una pestaña o ventana, el historial de la sesión se almacena en el disco.

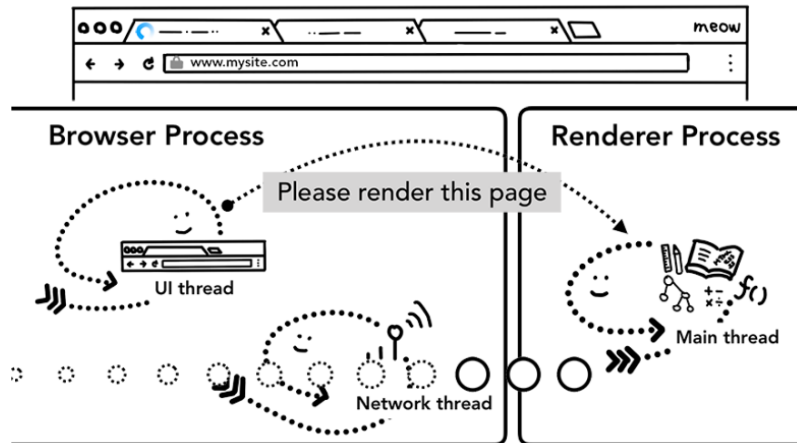


Figura 5. IPC entre el proceso navegador y los procesos del renderizador, solicitando renderizar la página.

- **Paso 5. Carga inicial completa:**

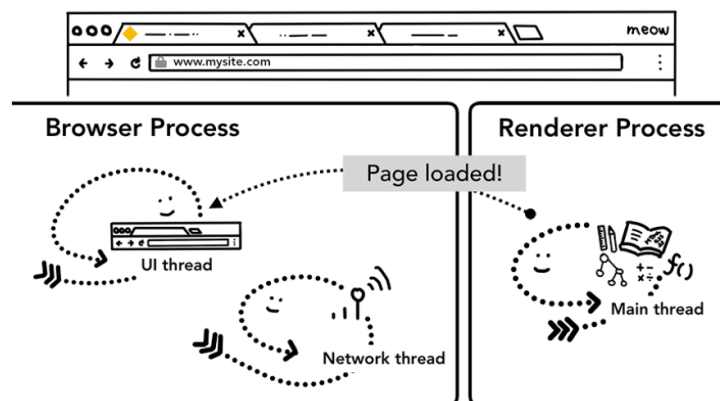


Figura 6. IPC desde el renderizador al proceso del navegador para notificar que la página se ha cargado.

Al finalizar la navegación, el proceso de renderizado continúa cargando recursos y renderiza la página. Una vez que el proceso de renderizado "finaliza", envía un IPC de vuelta al proceso del navegador. En este punto, el hilo de la interfaz de usuario detiene el control giratorio de carga en la pestaña.

Hemos puesto "finaliza" entre comillas porque el código JavaScript del lado del cliente aún podría cargar recursos adicionales y generar nuevas vistas después de este punto.

Esta parte, la parte de renderizado, es una de las que más nos interesa en nuestro módulo, por ello, la vamos a ver con mayor detenimiento:

DENTRO DEL PROCESO DE RENDERIZADO

Los procesos de renderizado manejan contenidos web y son los responsables de todo lo que sucede dentro de la pestaña.

En un proceso de renderizado, el hilo principal maneja la mayor parte del código que se envía al usuario. A veces, partes del código JavaScript son manejadas por **subprocesos de trabajo**. Los **subprocesos de composición y ráster** también se ejecutan dentro de los procesos de un renderizador para representar una página de manera eficiente y sin problemas.

El trabajo principal del proceso de renderizado es convertir HTML, CSS y JavaScript en una página web con la que el usuario pueda interactuar.

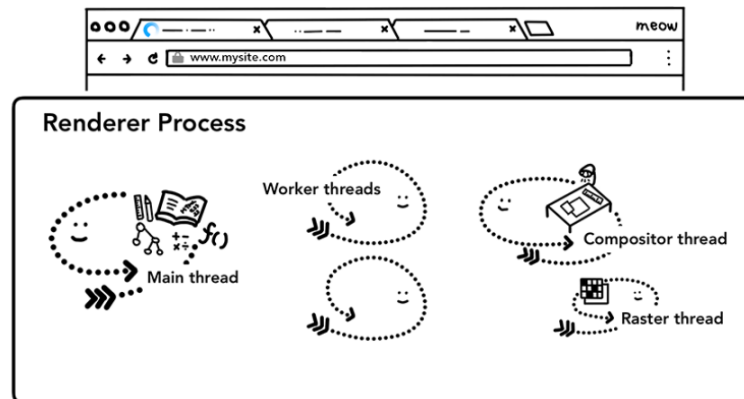


Figura 7. Proceso de renderizado con un subproceso principal, de trabajo, compositor y ráster

- Paso 1. Construcción de un DOM:** Cuando el proceso de renderizado recibe un mensaje para una navegación y empieza a recibir datos HTML, el hilo principal comienza a analizar (parsing) la cadena de texto (HTML) y convertirlo en un **Modelo de Objetos del Documento (DOM)**. El DOM es la representación interna de la página de un navegador, así como la estructura de datos y la API con la que el desarrollador web puede interactuar a través de JavaScript. El análisis de un documento HTML (parsing) en un DOM está definido por el [estándar HTML](#) .
- Paso 2. Carga de subrecursos:** Un sitio web suele utilizar recursos externos como imágenes, CSS y JavaScript. Esos archivos deben cargarse desde la red o localmente. El hilo principal *podría* solicitarlos uno por uno a medida que los encuentra mientras analiza (parsing) para construir un DOM, pero para acelerar el procedimiento, el "escáner de precarga" se ejecuta al mismo tiempo. Si hay etiquetas como **** o **<link>** en el documento HTML, el escáner de precarga envía solicitudes al hilo de red en el proceso del navegador.

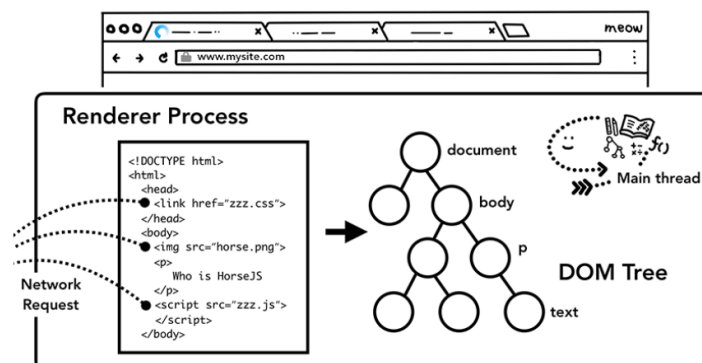


Figura 8. El hilo principal analizando HTML y construyendo el árbol del DOM.

Cuando el analizador HTML (HTML parser) encuentra una etiqueta `<script>`, detiene el análisis del documento HTML y tiene que cargar, analizar y ejecutar el código JavaScript mediante el **interprete de JavaScript** (normalmente en un worker threads). ¿Por qué? porque JavaScript puede cambiar la forma del documento usando instrucciones como `document.write()` que cambia toda la estructura DOM. Esta es la razón por la que el HTML parser tiene que esperar a que se ejecute JS antes de poder reanudar el análisis del documento HTML.

Se puede modificar este flujo agregando los atributos `async` o `defer` a la etiqueta `<script>`. Así, el navegador carga y ejecuta el código JavaScript de forma asincrónica y no bloquea el análisis. También puede usar los **módulos JavaScript**. Otra opción es usar `<link rel="preload">` para informar al navegador que el recurso es definitivamente necesario para la navegación actual y que le gustaría descargarlo lo antes posible.

- **Paso 3: Programando el estilo:** Tener un DOM no es suficiente para saber cómo se vería la página porque podemos diseñar elementos de página en CSS. El hilo principal analiza el código CSS y determina el estilo programado para cada nodo DOM, es decir qué tipo de estilo se aplica a cada elemento según los selectores de CSS.

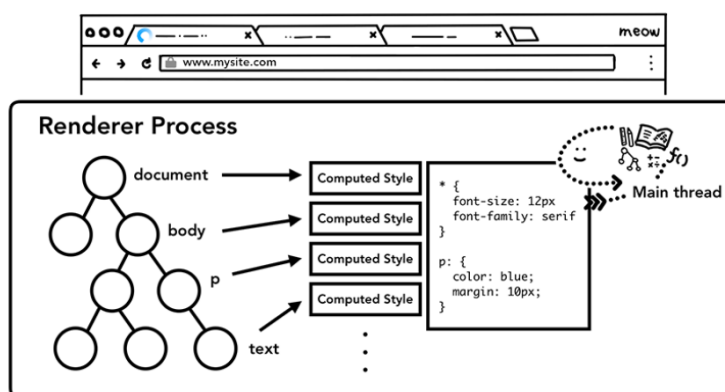


Figura 9. El hilo principal parseando CSS para añadir los estilos

Incluso si no se proporciona ningún CSS, cada nodo DOM tiene un estilo programado. La etiqueta `<h1>` se muestra más grande que la etiqueta `<h2>` y se definen márgenes automáticamente para cada elemento. Esto se debe a que el navegador tiene una hoja de estilo predeterminada.

- **Paso 4. Diseño (Layouts):** En este punto el proceso de renderización conoce la estructura de un documento y los estilos de cada nodo, pero eso no es suficiente para renderizar una página. Imagina que estás tratando de describirle un cuadro a tu amigo por teléfono. "Hay un gran círculo rojo y un pequeño cuadrado azul" no es información suficiente para que su amigo sepa cómo se vería exactamente el dibujo..

El diseño es un proceso para encontrar la geometría de los elementos. El hilo principal recorre el DOM y los estilos programados y crea el árbol de diseño (Layout Tree) que tiene información como las coordenadas y los tamaños de los cuadros delimitadores. El árbol de diseño puede tener una estructura similar al árbol DOM, pero solo contiene información relacionada con lo que está visible en la página.

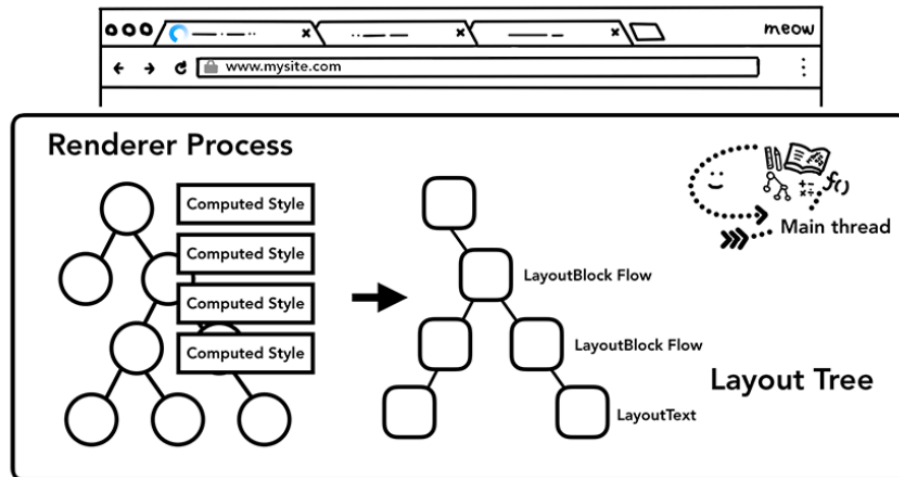


Figura 10. El hilo principal que pasa por el árbol DOM con estilos calculados y produce un árbol de diseño

- **Paso 5 Pintura (Paint).** Tener un DOM, estilo y diseño todavía no es suficiente para renderizar una página. Digamos que está intentando reproducir una pintura. Conoces el tamaño, la forma y la ubicación de los elementos, pero aún tienes que juzgar en qué orden los pintas.

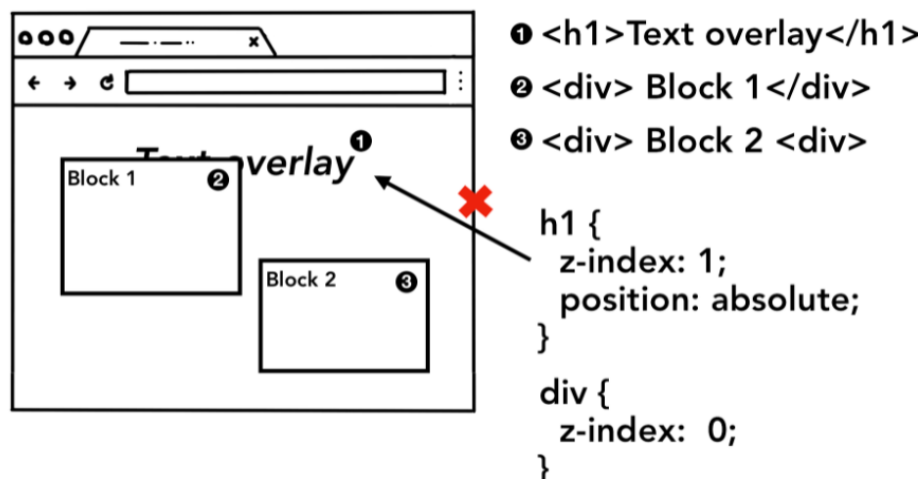


Figura 11. Elementos de la página que aparecen en el orden del HTML, lo que da como resultado una imagen renderizada incorrecta

- **Paso 6: Composición:** Ahora que el navegador conoce la estructura del documento, el estilo de cada elemento, la geometría de la página y el orden de pintado, ¿cómo dibuja una página? Convertir esta información en píxeles en la pantalla se denomina rasterización. Quizás una forma ingenua de manejar esto sería rasterizar partes dentro de la ventana gráfica. Si un usuario se desplaza por la página, se mueve el marco rasterizado y se completan las partes faltantes rasterizando más. Así es como Chrome manejó la rasterización cuando se lanzó por primera vez. Sin embargo, el navegador moderno ejecuta un proceso más sofisticado llamado composición.

La composición es una técnica para separar partes de una página en capas, rasterizarlas por separado y componerlas como una página en un hilo separado llamado hilo del compositor. Si ocurre un desplazamiento, dado que las capas ya están rasterizadas, todo lo que tiene que hacer es componer un nuevo marco.

3. Lenguajes de programación en entorno cliente

A pesar de que han existido (applets de Java, ActionScript para Adobe Flash) y existen (ASP.NET de Microsoft) multitud de tecnologías y lenguajes de programación, en la actualidad el desarrollo web en entorno cliente se basa en tres pilares fundamentales:

- **HTML:**
 - No es un lenguaje de programación, sino el lenguaje de marcas de hipertexto (HyperText Markup Language) más usado en la Web para mostrar el contenido de un documento.
 - Se basa en el uso de un sistema de etiquetas cerrado aplicado a un documento de texto.
 - No necesita ser compilado, sino interpretado por el navegador a medida que se avanza en el documento.
 - Permite: organizar texto y objetos, crear listas y tablas y, obviamente, permitir los hipervínculos (esencia de la Web).
 - La versión actual es HTML5.
- **CSS:**
 - Hojas de estilos en cascada (Cascade Style Sheets)
 - Es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado como HTML.
 - Las hojas de estilo sirven para separar el formato que se quiere dar a la página de la estructura de ésta y demás instrucciones
 - La versión actual es CSS3
- **JavaScript:**
 - Lenguaje de programación de scripting (interpretado por el navegador) y embebido/incluido en un documento HTML.
 - Se define como orientado a prototipos, débilmente tipado y con características dinámicas.
 - Permite mejoras en la interfaz del usuario y la creación de páginas dinámicas.
 - Sintaxis similar a C, aunque adopta nombres y convenciones propias de Java (aunque no tiene ninguna relación con este último lenguaje).
 - El estándar se especifica en: [ECMAScript](#).
 - Todos los navegadores modernos lo interpretan.

JavaScript es el lenguaje que nos permite dotar de interactividad y cambiar el comportamiento a los elementos que componen una página web, por tanto, será el centro de interés de este módulo.

Además, los frameworks front-end que se suelen usar en las empresas de desarrollo web, principalmente React, Angular y Vue, están basados en JavaScript o TypeScript (superconjunto de JavaScript), con lo cual, conocer las bases y fundamentos de este lenguaje de scripting es imprescindible.

3.1. Características de los lenguajes de script (guiones).

Los scripts surgieron como fragmentos de código que realizaban ciertas tareas o rutinas concretas. En los sistemas operativos, los scripts se usan para automatizar tareas y siempre van a ser ejecutadas por un intérprete de comandos.

Gracias a los scripts, se utiliza un sistema host (como puede ser el navegador) para integrar pequeños bloques de código que aporten nuevas funcionalidades dotando así de un aspecto más dinámico a las aplicaciones web.

La potencia de estas funcionalidades ha ocasionado que los scripts ya no sean pequeños trozos de código sino que conformen auténticos programas. En ocasiones, los scripts se incrustan dentro de otros programas o códigos (como JS se integra dentro de un HTML).

En general:

- los lenguajes scripts son interpretados y los lenguajes de programación compilados.
- los scripts usan componentes existente y los LP parten desde cero.
- los scripts se ejecutan dentro de otro programa y los LP lo hacen de forma independiente.

Ejemplos típicos son:

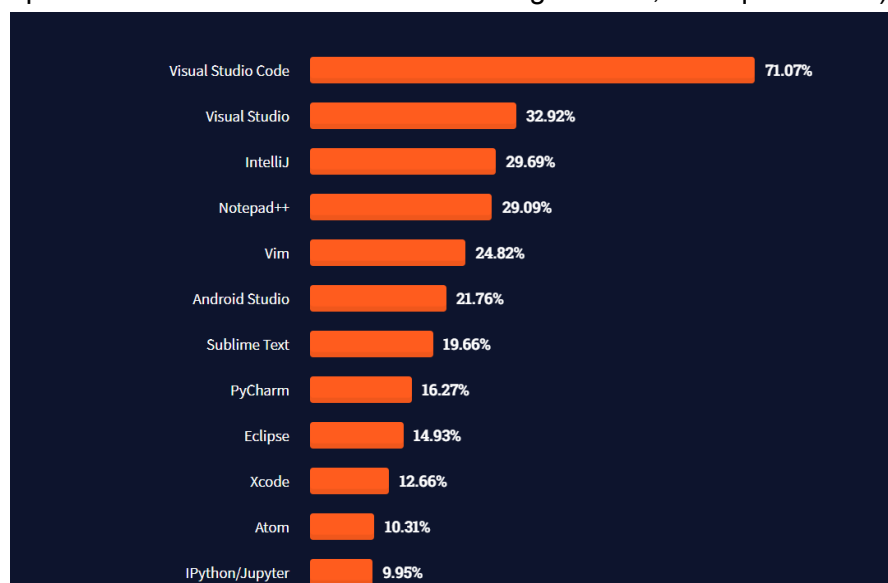
- Lenguajes de programación: C, C++, C#, Java, Pascal...
- Lenguajes de scripting: JavaScript, PHP, Python, Shell...

4. Herramientas y utilidades de programación

4.1. Visual Studio Code

Antes de empezar a trabajar con el código JavaScript, será oportuno configurar un entorno de trabajo que se adecue a nuestras necesidades. Al tratarse JS de un lenguaje de scripting, para empezar a programar bastaría con un utilizar un editor de texto, pero un entorno de trabajo profesional exige aprovechar las ventajas que ofrece un entorno de desarrollo integrado (IDE).

Según la encuesta de StackOverflow de 2021, el IDE más utilizado por los desarrolladores es VS Code (aunque estrictamente sea un editor de código fuente, más que un IDE):



Fuente: <https://insights.stackoverflow.com/survey/2021#most-popular-technologies-new-collab-tools-prof>



Además de ser el más utilizado, VS Code es gratuito y open source, relativamente ligero, tiene un gran soporte de extensiones, implementa CLI (interfaz de línea de comandos) y posee integración nativa de Git.

Por ello, hemos decidido usar VSCode en el módulo, pero no es obligatorio su uso, si estás habituado a utilizar cualquier otro, no existe ningún inconveniente en que sigas usándolo. Los pasos a seguir serán similares a estos:

- Paso 1 Instalación: Te puedes descargar VS code desde su página oficial, eligiendo la plataforma adecuada: <https://code.visualstudio.com/>
 - Paso 2 Configuración:
 - Instalar idioma
 - Seleccionar tema: Claro, oscuro, contraste..
 - Configuración: Ctrl+,
 - Paso 3: Instalación de node.js
 - Binarios:
 - Descargar de la página oficial: <https://nodejs.org/es>
 - Crear directorio: `sudo mkdir /usr/local/lib/nodejs`
 - Extraer y mover: `sudo tar -xJvf node-v14.17.5-linux-x64.tar.gz -C /usr/local/lib/nodejs`
 - Modificar bashrc: `nano ~/.bashrc` (también se puede hacer en ~/.profile) incluyendo línea: `export PATH=/usr/local/lib/nodejs/node-v14.17.5-linux-x64/bin:$PATH`
 - Ejecutarlo: `./bashrc`
 - Comprobar versión: `node -v` y `npm -v`
 - Apt:
 - `sudo apt update`
 - `sudo apt install nodejs`
 - `node -v`
 - `sudo apt install npm`
 - `npm -v`
 - Paso 3 Instalación de extensiones:
 - Linter (ESLint): Revisa el formato y las partes de código muerto o no usado. Ejemplos: variables o funciones sin usar. Variables que no se modifiquen pueden declararse como constantes, etc. Es configurable a través de reglas. Además de la extensión habría que:
 - Previamente, tener un `package.json` (`npm init`)
 - Instalarlo: `npm install eslint -D`
 - Inicialo: `./node_modules/.bin/eslint --init` o `npx eslint --init` (últimas versiones de npm)
 - Ver errores: `npx eslint .`
 - Corregir (si puede) esos errores: `npx eslint . --fix`
- Si usamos extensión ESLint:
- En settings.json de VS Code escribir para que corrija al guardar:


```
"editor.codeActionsOnSave": {
  "source.fixAll.eslint": true
}
```



- Prettier: Da formato solo. Se recomienda no usar reglas que modifiquen su formato. Puede tener incompatibilidades con ESLint. Para trabajar juntos, instalar configuración especial de ESLint: `npm install eslint-config-prettier -D` y poner 'prettier' como valor final del extends de `.eslintrc.js` antes de instalar:

- Instalar: `npm install prettier -D`
- Crear fichero de configuración vacío: `.prettierrc.json`
- Buscar errores: `npx prettier . --check`
- Corregir errores: `npx prettier . --write`

Para no hacerlo manualmente en consola, integramos la extensión:

- Instalar extensión Prettier
- Establecer por defecto el formateador de prettier en `settings.json`:
`"editor.defaultFormatter": "esbenp.prettier-vscode"`

- JavaScript (ES6) Code Snippets: Contiene atajos a los bloques de código más usados en JavaScript.
- LiveServer: Lanza un servidor local de desarrollo
- VSCode-icons o Material Icon Theme: Ayuda visual para identificar las diferentes carpetas con iconos, por defecto suele traer alguno (seti).
- Bracket Pair Colorized 2: Da el mismo color a pares de paréntesis y llaves. Última versión de VS Code ya trae la opción: `settings.json` → `"editor.bracketPairColorization.enabled": true`

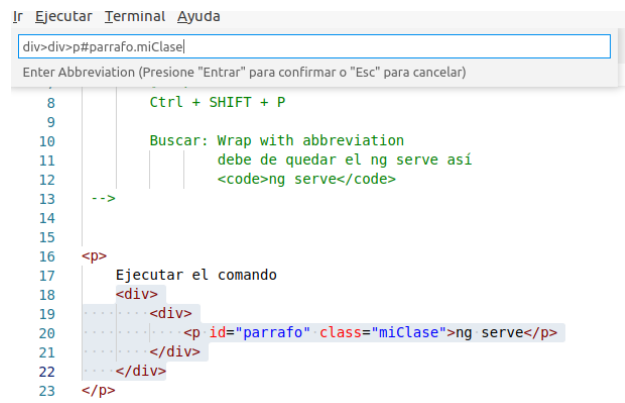
● Atajos interesantes:

Display

- Ctrl+Shift+P: Abre la paleta de comandos
- Ctrl+J: Abre terminal
- Ctrl+Tab: Navegar entre pestañas
- Ctrl+B: Abrir/Cerrar menú lateral

Edición

- Shift+Tab: Quita tabulación
- Ctrl+/ (Ctrl + Shift + A): Comenta líneas seleccionadas (Ctrl + ¢)
- Alt+Shift+A: Comenta parte de una línea
- F12: Nos lleva hasta la definición de una función, variable...
- Ctrl+F12: Muestra en una ventana la definición
- Ctrl+Shift+L: Selecciona todas las ocurrencias de una selección
- Ctrl+Shift+K: Borra líneas completas
- Envolver líneas seleccionadas: Ctrl+Shift+P → > → wrap abbreviation → escribir etiqueta. Ejemplo:



Nota: Se pueden crear accesos directos buscando el comando en Ctrl+K Ctrl+S



Movimiento de líneas

- Ctrl+G: Ir a un determinado número de línea
- Ctrl + ↑ ↓: Desplaza verticalmente el código entero, dejando el cursor en el mismo sitio
- Alt + ↑ ↓: Desplaza verticalmente una línea de código.
- Shift + ↑ ↓: Selecciona líneas de código
- Alt + Z: Justifica todo el código en la pantalla (sin barra de desplazamiento horizontal)

Cursor

- Ctrl + Shift + ↑ ó ↓: Crea cursores arriba o abajo

Ejemplo:

```

2      Objetivo:
3          Crear múltiples cursores
4          Añadir la clase de estilo igual a la contenida en el span.
5          Ver objetivo final
6
7      Tips:
8          Ctrl + Shift + ↑ / ↓ //Para generar los cursores en cada línea
9          Ctrl + Shift + →      //Para desplazar la selección y seleccionar solo los colores
10     -->
11
12     <span>amarillo</span>
13     <span>rojo</span>
14     <span>verde</span>
15     <span>naranja</span>
16     <span>morado</span>
17     <span>negro</span>
18     <span>blanco</span>
19
20     <!-- Objetivo final -->
21     <span class="amarillo">amarillo</span>
22     <span class="rojo">rojo</span>
23     <span class="verde">verde</span>
24     <span class="naranja">naranja</span>
25     <span class="morado">morado</span>
26     <span class="negro">negro</span>
27     <span class="blanco">blanco</span>

```

- Ctrl + D: Seleccionar siguiente ocurrencia

Ejemplo:


```

1      /*
2      Objetivo:
3          Crear múltiples cursores
4          Y usarlo para editar un arreglo de días de la semana
5
6      Tips:
7          Ctrl + Alt+ ↑ / ↓
8
9      */
10
11     // Crear un arreglo con los días de la semana
12
13     lunes
14     martes
15     miércoles
16     jueves
17     viernes
18     sábado
19     domingo
20
21
22     // Objetivo final
23
24     const diasDemo = ['lunes', 'martes', 'miércoles', 'jueves', 'viernes',
25                       'sábado', 'domingo'];
26
27

```

4.2. Utilidades de desarrollo del navegador (<https://developer.chrome.com/docs/devtools/>)

Los navegadores más utilizados suelen integrar un conjunto de herramientas para desarrolladores. En Firefox y Chrome aparecen pulsando F12 o CTRL+MAY+I. Estas herramientas son:

- Modo Responsivo: 
- Pestaña Elements: Se visualizan los elementos del DOM. Al seleccionar un elemento, la variable \$0 hará referencia a este. Es editable (además si ponemos a una etiqueta atributo *contentEditable* se puede editar en línea). A la derecha aparece otro panel con las características de los elementos seleccionados:
 - Styles: Propiedades CSS. Es editable. (User Agent: Identificador Interno del navegador)
 - Computed: Valores definitivos de estilo que muestra, también visualiza la parte de content-padding-border-margin del elemento seleccionado
 - Layout: Bordes para ayudar en el diseño
 - Event Listeners: Eventos de JS.
- Pestaña Consola: Permite ejecutar comandos JS. Además se suele usar para mandar mensajes desde el código JS: console.log, table, dir, time (timeEnd)... Las configuraciones de esta pestaña permiten:
 - Preserve Log: No borra la consola al actualizar
- Pestaña Sources: Aparecen los dominios a los que se está haciendo peticiones y los recursos que se descargan (archivos html, js...). Pulsando el icono de las llaves formatea de manera más legible el código.
 - Debugger: Incluye un depurador propio (se puede activar con debugger en js)
- Pestaña Network: Muestra las peticiones y sus características. Si se desactiva la caché seguirá desactivada al cerrar las herramientas.
- Pestaña Performance: Muestra el rendimiento de una página web, es decir, cómo y cuánto tarda en cargarse. Marcar Screenshots. Web Vitals: Se usan para SEO. Memory muestra el consumo de memoria.
- Pestaña Memory
- Pestaña Application: Útil para visualizar las cookies, localStorage, sessionStorage...
- Pestaña Security: Aspectos relacionados con el certificado de seguridad (HTTPS)
- Pestaña Lighthouse: Informe resumen de aspecto de tu web como rendimiento, accesibilidad, UX...
 - Cumulative Layout Shift: Tiene en cuenta si los elementos van cambiando de posición conforme se carga la web, pudiendo provocar clicks involuntarios

Existen muchas más herramientas que nos pueden ayudar a desarrollar con JS: codepen.io, coding ground...

5. Integración de código JS con HTML

Fundamentalmente existen dos maneras de incorporar JS en nuestro proyecto:

5.1. Código JS dentro del HTML

Se incluye el código dentro de las etiquetas `<script></script>`. Ya no es necesario incluir: `<script type= "text/javascript">` puesto que el navegador entiende por defecto que será JS el código que hay dentro de esta etiqueta.

El código se puede colocar dentro de la etiqueta head como dentro de la etiqueta body. Lo lógico es colocarlo siempre en el mismo sitio para facilitar la legibilidad y mantenimiento. Además, hay que tener en cuenta los problemas que suponen cargar el script en el head ya que se puede estar haciendo referencia a elementos que aún no están cargados.

5.2. Código JS en archivos separados

Esta es la alternativa más eficaz, ya que sigue el diseño por capas. Esta es la forma en la que se suele encontrar el código JS en un proyecto profesional. Se suele incluir en el head de la siguiente manera:

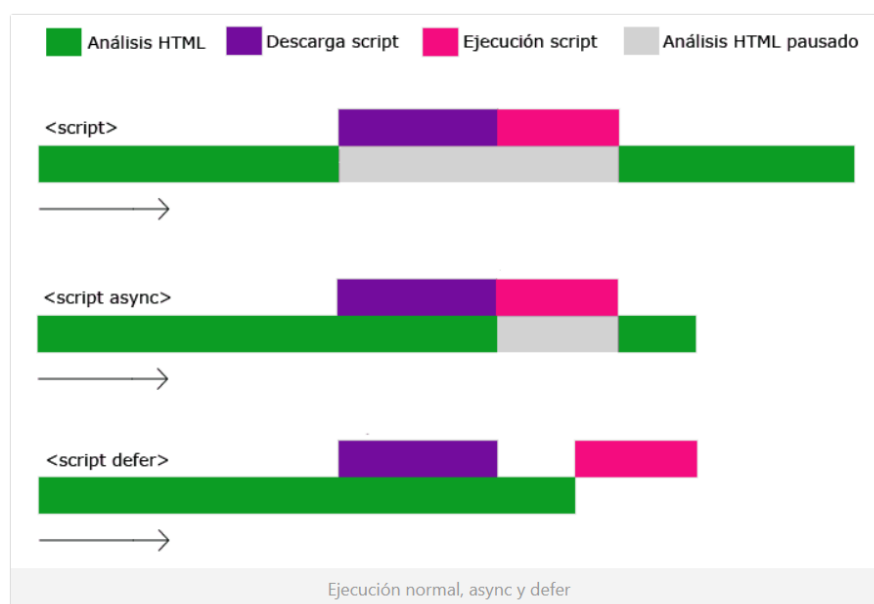
- `<script src= "../script/codigo.js"></script>`

Se está indicando que el código JS se encuentra en un archivo aparte llamado codigo.js dentro de la carpeta script que se encuentra en el directorio del index.html.

Las ventajas de trabajar por capas, es decir, con archivos separados son:

- Código HTML independiente
- Código más fácil de mantener
- Carga más rápida.

El inconveniente también suele ser que se ejecute el código JS antes de que el DOM esté totalmente parseado. Por eso se suele agregar los atributos `async` y preferiblemente `defer`:



Fuente: <https://cybmeta.com/diferencia-async-y-defer>



- **<script>** (normal): el análisis HTML se detiene, se descarga el archivo (si es un script externo), se ejecuta el script y después se reanuda el análisis HTML.
- **<script async>**: el script se descarga de forma asíncrona, es decir, sin detener el análisis HTML, pero una vez descargado, si se detiene para ejecutar el script. Tras la ejecución se reanuda el análisis HTML. Sigue existiendo un bloqueo en el renderizado pero menor que con el comportamiento normal. **No se garantiza la ejecución de los scripts asíncronos en el mismo orden en el aparecen en el documento.**
- **<script defer>**: el script se descarga de forma asíncrona, en paralelo con el análisis HTML, y además su ejecución es diferida hasta que termine el análisis HTML (pero antes de lanzar el evento DOMContentLoaded). No hay bloqueo en el renderizado HTML. La ejecución de todos los scripts diferidos se realiza en el mismo orden en el que aparecen en el documento.

Representación gráfica: <https://manzdev.github.io/script-type/>

6. Posibilidades de JS

Como ya se ha comentado alguna vez en clase, JS es, dentro del paradigma de las 3 capas, la capa encargada de dotar de interactividad y comportamiento a nuestras páginas webs (HTML, la estructura y CSS, el estilo).

Dedicaremos gran parte del curso a conocer todos los recursos que ofrece JS, pero para poder hacernos una idea de lo que se puede hacer con JS vamos a ver brevemente alguna de sus posibilidades:

6.1. Modificación de contenido de una página web.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Modificando el código HTML</h1>
  <p id="miParrafo">Párrafo Original</p>
  <button onclick="document.getElementById('miParrafo').innerHTML =
  'Párrafo cambiado'">Cambia HTML</button>
</body>
</html>
```

6.2. Cambiar atributos de elementos HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Cambio de imágenes con JS</h1>
  <h2>Pulsa la imagen para cambiarla</h2>
  

  <script>
    function cambiarImagen() {
      const imagen=document.getElementById("miImagen");
      if (imagen.src.match("piqsels")) {
        imagen.src="https://p1.pxfuel.com/preview/86/583/812/code-computer-css-it.jpg";
      }else{
imagen.src="https://p2.piqsels.com/preview/792/452/926/code-text-screen-webcomponent.jpg";
      }
    }
  </script>
</body>
</html>
```

6.3. Cambiar el estilo

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Cambio de estilo con JS</h1>
  <p id="miParrafo">TEXTO A MODIFICAR</p>
  <button onclick="cambiarEstilo()">Cambiar</button>

  <script>
    function cambiarEstilo() {
      const parrafo=document.getElementById("miParrafo");
      parrafo.style.fontSize="25px";
      parrafo.style.color="red";
    }
  </script>
</body>
</html>
```

7. Bibliografía

- <https://platzi.com/blog/que-es-frontend-y-backend/>
- <https://www.html5rocks.com/es/tutorials/internals/howbrowserswork/>
- <https://developers.google.com/web/updates/2018/09/inside-browser-part2>
- <https://gs.statcounter.com/>
- <https://www.ecma-international.org/publications-and-standards/standards/ecma-262/>
- <https://insights.stackoverflow.com/survey/2021>
- <https://code.visualstudio.com/docs/getstarted/tips-and-tricks>