



Selección de arquitecturas y herramientas de programación

Objetivos

- Conocer las fases del ciclo de desarrollo del software.
- Exponer la diferencia entre los lados cliente y servidor.
- Analizar la arquitectura de tres niveles de las aplicaciones web.
- Concretar las tecnologías que intervienen en el lado cliente y justificar el aprendizaje de JavaScript como lenguaje de la web.
- Enumerar las herramientas necesarias para el desarrollo básico en el lado cliente y las deseables para desarrollos más complejos.

Contenidos

- 1.1. Desarrollo de software
- 1.2. Lado cliente y lado servidor
- 1.3. Herramientas para el desarrollo en el lado cliente

Introducción

El mundo de la informática experimentó un crecimiento exponencial con la llegada de internet. El acceso a la información se democratizó y se extendió a todos los rincones del planeta. Tal fue la importancia de esta innovación tecnológica que se popularizó la expresión «si no estás en la web, no existes».

Pero los sitios web no nacen por generación espontánea, necesitan profesionales que los desarrollen e infraestructura que los soporten. Desde aquellos primeros días hasta hoy el cambio y la especialización han sido tan radicales que hasta el concepto de desarrollo web parece demasiado amplio para abarcar todas las tecnologías que intervienen. Por todo ello, se hace necesario situar el contexto, conocer la foto global y poner el foco en aquellos conceptos imprescindibles para comenzar a caminar en la apasionante tarea del desarrollo web. De eso se encarga esta primera unidad.

1.1. Desarrollo de software

De forma resumida puede definirse el desarrollo de aplicaciones como el proceso de creación de un programa informático (por tanto, *software*) o un conjunto de programas para realizar las diferentes tareas que cubren una necesidad. Esta necesidad puede haberse generado en cualquier parte de una organización: mostrar el detalle de un producto, aplicar descuentos, generar informes, consultar información externa, presentar resultados de operaciones, o cualquier otra tarea imaginable. Sin embargo, a pesar de la heterogeneidad de su naturaleza, todas estas necesidades comparten una necesidad común: automatización y aumento de la eficiencia, dos estrategias que generan ventajas competitivas para las empresas.

Cada proceso de creación de aplicaciones sigue los mismos pasos:

- 1. Planificación y análisis de requisitos.** Es una de las etapas más importantes. La realizan los miembros más experimentados del equipo junto con el cliente. Esta información se utilizará para planificar el enfoque básico del proyecto y estudiar su viabilidad económica, operativa y técnica. Además, también se identifican los riesgos asociados para minimizarlos todo lo posible.
- 2. Definición de requisitos.** Definir y documentar claramente los requisitos y obtener la aprobación del cliente. Serán estos requisitos los que se diseñarán e implementarán durante el ciclo de vida de desarrollo del software.
- 3. Diseño de la arquitectura.** Se aborda la solución desde diferentes puntos de vista hasta encontrar el enfoque que se considera óptimo. Este enfoque de diseño debe definir claramente todos los módulos junto con su comunicación y representación del flujo de datos con módulos externos (si los hubiera). El diseño interno de todos los módulos debe estar definido con el más mínimo detalle. Además, debe incluir unos estándares de calidad que se pretenden alcanzar.

- 4. Construcción o implementación.** Si las fases anteriores se realizaron correctamente, la generación de código debería lograrse sin mucho esfuerzo. Aquí, se utilizan herramientas de programación como compiladores, intérpretes, depuradores, etc. La elección del lenguaje de programación estará fuertemente vinculado al tipo de software que se esté desarrollando.
- 5. Pruebas.** Los defectos del producto se informan, rastrean, reparan y vuelven a probar, hasta que el producto alcanza los estándares de calidad definidos en el diseño de la arquitectura.
- 6. Despliegue y mantenimiento.** Se despliega la aplicación en producción, se corregen los defectos detectados por el contexto de ejecución y se realizan todas las modificaciones necesarias para que opere de forma eficiente.

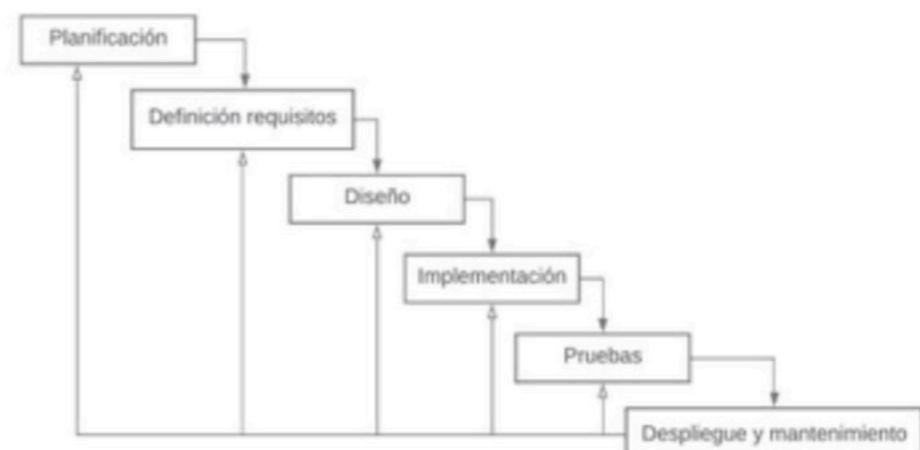


Figura 1.1. Ciclo de desarrollo de software tradicional en cascada.

Este es el modelo común, aunque según las políticas de cada organización las fases pueden granularse o compactarse para ajustarse mejor a las particularidades de sectores concretos.

También es importante señalar que unas fases consumirán más tiempo y recursos que otras dependiendo del tipo de aplicación que se esté desarrollando.

Actividad propuesta 1.1

Ciclos de vida del software

Busca información sobre estos cinco ciclos de vida de un software y compara sus diferencias:

- Modelo en cascada.
- Modelo iterativo.
- Modelo de desarrollo incremental.
- Modelo en espiral.
- Modelo de prototipos.

1.1.1. Tipos de aplicaciones

Las clasificaciones de los tipos de aplicaciones que se pueden desarrollar han cambiado mucho a lo largo del tiempo. Si bien, la última clasificación que se ha venido utilizando atendía a los tipos de dispositivos para los que estaban diseñadas (escritorio, web y móvil), en los últimos años se ha especializado enormemente esta tarea centrándose el debate en el tipo de desarrollos, puesto que en la actualidad una sola aplicación puede servir perfectamente para todo tipo de dispositivos. Veamos algunos de los enfoques más utilizados para entender la foto global del desarrollo en la que se centrará esta obra:

- **Desarrollo de aplicaciones personalizadas.** Cuando el software comercial no cumple con los requisitos específicos de una organización, la mejor opción es personalizarlo.
- **Desarrollo rápido de aplicaciones.** Sigue un enfoque incremental. Se pueden desarrollar simultáneamente los módulos individuales de la aplicación para acortar los tiempos.
- **Desarrollo de aplicaciones low-code.** Las plataformas de desarrollo de poco código incluyen interfaces visuales intuitivas que facilitan la creación y el lanzamiento de aplicaciones incluso para no programadores. Se pueden incluir y modificar fragmentos de código a golpe de ratón según la lógica de negocio, permitiendo incluso diseñar flujos de trabajo automatizados.
- **Desarrollo de aplicaciones móviles.** Destinado a la creación de aplicaciones que se ejecutan en cualquier plataforma móvil.
- **Desarrollo de aplicaciones de bases de datos.** Estas aplicaciones están diseñadas para recopilar, organizar y administrar información de manera eficiente. Se utilizan para ordenar datos por criterios, realizar cálculos, crear informes y compartir información.
- **Desarrollo de aplicaciones empresariales.** Son un tipo de software desarrollado para atender a organizaciones a gran escala. Admiten funciones de nivel empresarial, como el almacenamiento masivo de datos y la automatización de procesos comerciales complejos.
- **Desarrollo de aplicaciones web.** Son software desarrollado para un propósito específico, alojado en un servidor web. Se puede acceder a ellas utilizando cualquier dispositivo con acceso a internet, como teléfonos, ordenadores o tabletas. Algunos ejemplos comunes se encuentran en sitios de banca *online*, tiendas *online* o las redes sociales.

1.1.2. Desarrollo de aplicaciones web

El desarrollo web es la construcción y el mantenimiento de sitios web. Por decirlo de una forma sencilla, es el trabajo que conduce a que un sitio web se vea bien, funcione rápido y con una experiencia de usuario satisfactoria.

Los desarrolladores lo consiguen utilizando una enorme variedad de lenguajes de programación, cuya elección, como ya se avanzó, depende en gran medida de las tareas que realizan y de las plataformas en las que trabajan.

Las habilidades para el desarrollo web tienen una gran demanda en todo el mundo y están muy bien pagadas en comparación con otro tipo de sectores técnicos, lo que convierte este trabajo en una excelente opción profesional.

1.2. Lado cliente y lado servidor

El campo del desarrollo web generalmente se divide en **front-end** (el lado del usuario o lado del cliente) y **back-end** (el lado de la máquina o lado del servidor). Veamos las características de cada lado:

- **Lado cliente.** Se trata de todo aquello que el usuario puede ver, con lo que puede interactuar y experimentar. El objetivo de estos desarrolladores es programar las partes del sitio web que son visibles para el usuario y que se ejecutan en el navegador. Desarrollan los diseños de la interfaz de usuario y su experiencia, que son los elementos clave para dar vida a la interfaz.
- **Lado servidor.** Las acciones realizadas por el usuario son analizadas, recuperadas y devueltas por el back-end a través de los programas almacenados y ejecutados en el servidor que los aloja. El trabajo de estos desarrolladores incluye vincular todos los aspectos del front-end entre sí y con las bases de datos desde donde extraen los datos que aparecerán en el front-end. El lado servidor es muy importante porque es donde está programada la lógica de negocio de la aplicación.



Figura 1.2. Dos clientes consumiendo datos de un servidor.

En la relación que muestra la Tabla 1.1 se puede ahondar aún más en los detalles de ambos lados.

En los últimos años, como consecuencia de la incorporación de tecnologías que trabajan en ambos lados, han proliferado multitud de perfiles técnicos de pila completa o *full stack*. Se refiere a desarrolladores que son capaces de crear tanto la parte frontal como la parte trasera de una aplicación web, fundamentalmente con tecnologías construidas sobre JavaScript.

Tabla 1.1. Principales diferencias entre el desarrollo de los lados cliente y servidor

	Lado cliente	Lado servidor
Definición	Implica la implementación efectiva de los componentes visuales de una aplicación web.	Implica la implementación efectiva de funcionalidades de una aplicación web, donde se incluyen el acceso a datos, la administración de servidores, etcétera.
Habilidades requeridas	HTML, CSS, SASS, JavaScript...	Python, Ruby, Java, PHP...
Independencia	No puede funcionar de forma independiente excepto en el caso de sitios estáticos.	Funciona de forma independiente respecto del <i>front-end</i> .
Objetivo	Garantizar que todos los usuarios puedan acceder a la aplicación y que siga respondiendo en todos los dispositivos.	Garantizar que la aplicación se ejecute en todos los casos, sea escalable y funcione de manera eficiente con baja latencia y sin fallos.
Equipo de desarrollo	Su trabajo es diseñar y desarrollar la apariencia e interactividad de la aplicación en función de la entrada de usuario.	Su trabajo es proporcionar datos al <i>front-end</i> , vincular páginas, brindar seguridad y soporte a los usuarios.
Frameworks utilizados	AngularJS, React, vue.js...	Django, Flask, CakePHP, Laravel, Ruby on Rails...
Habilidades adicionales	Una buena comprensión del diseño de UI y UX.	Razonamiento lógico y resolución de problemas.

Nota técnica

Se llama **framework** al conjunto de herramientas y librerías software que proporcionan numerosas utilidades para el desarrollo de aplicaciones web más escalables y sencillas de mantener, por lo que conducen a un importante ahorro de recursos.

UX (*User eXperience*, experiencia de usuario) y **UI** (*User Interface*, interfaz de usuario) son muy importantes en el desarrollo de aplicaciones web puesto que determinan la satisfacción de uso de una aplicación. Mientras que el segundo se centra en la interacción del usuario con la aplicación a través de los elementos diseñados para ello, el primero trata de que esta experiencia de uso sea lo más amigable, sencilla e intuitiva posible.



1.2.1. Arquitectura

Dejando a un lado las tecnologías concretas que intervienen en cada lado de una aplicación, es también muy importante conocer la arquitectura de una aplicación web típica.

Las aplicaciones web de hoy en día utilizan las tecnologías de ambos lados siguiendo una **three-tier architecture** o arquitectura de tres niveles. Se trata de la arquitectura de

software predominante para las aplicaciones cliente-servidor tradicionales. Organiza las aplicaciones en tres niveles informáticos lógicos y físicos: el nivel de presentación o interfaz de usuario; el nivel de aplicación, donde se procesan los datos; y el nivel de datos, donde se almacenan y gestionan los datos asociados a la aplicación.

El principal beneficio de esta separación es que cada nivel se ejecuta en su propia infraestructura, cada nivel puede ser desarrollado simultáneamente por un equipo de desarrollo independiente y puede actualizarse o escalarse según sea necesario sin afectar a los otros niveles. A continuación se verá cada nivel con un poco más de detalle:

- **Nivel de presentación.** Define la interfaz de usuario y la comunicación con la aplicación, donde el usuario final interactúa con la aplicación. Su objetivo es mostrar información y recopilar información del usuario. En el campo de la web, se ejecuta en el navegador y generalmente se desarrolla utilizando HTML, CSS y JavaScript. En el desarrollo web este nivel estaría representado como el servidor web.
- **Nivel de aplicación.** También conocido como nivel lógico o intermedio, es el corazón de la aplicación. La información recopilada en el nivel de presentación se procesa, a veces con la colaboración del nivel de datos, utilizando la lógica de negocio. Desde este nivel también se pueden añadir, modificar o eliminar datos del nivel de datos. Generalmente se desarrolla utilizando Python, Java, PHP o Ruby, y se comunica con el nivel de datos mediante llamadas a su API. En el desarrollo web este nivel estaría representado como el servidor de aplicaciones.
- **Nivel de datos.** También conocido como nivel de bases de datos o nivel de acceso a datos. Es donde se almacena y administra la información procesada por la aplicación. Este puede ser un sistema de gestión de base de datos relacional como MySQL, PostgreSQL, Oracle o Microsoft SQL Server, o un servidor de base de datos NoSQL como MongoDB, CouchDB o Cassandra. En el desarrollo web este nivel estaría representado como el servidor de bases de datos.

En una aplicación de tres niveles, toda la comunicación pasa por el nivel de aplicación. El nivel de presentación y el nivel de datos no pueden comunicarse entre sí.

**Figura 1.3.** Esquema conceptual con la lógica de petición y la respuesta asociada.

El usuario, a través del navegador, realiza una acción. Esta acción desencadena una petición http al servidor web. El servidor web necesita realizar cálculos de procesamiento

pertenecientes a la lógica de negocio, por lo que solicita al servidor de aplicaciones que realice esta tarea. Es posible que el servidor de aplicaciones necesite datos que están almacenados, por lo que solicita estos datos al nivel de datos. Una vez completado el flujo de solicitudes se comienza a construir la respuesta al usuario. Los datos llegan a los ficheros que los procesarán, y estos generarán ficheros de presentación que el servidor web mostrará al usuario en el navegador en un formato entendible.

Ahora que ya se ha descrito la foto global de todos los conceptos que rodean al desarrollo web, es el momento de centrarse en el objeto de esta obra: el lado cliente.

■ ■ ■ 1.2.2. Tecnologías del lado cliente

Como ya se ha descrito con anterioridad, el marco de ejecución de las aplicaciones web en este lado es el navegador. Por este motivo, para el desarrollo web en el entorno cliente es necesario dominar al menos estas tres tecnologías:

- **HTML** (*HyperText Markup Language*, lenguaje de marcado de hipertexto) es un lenguaje que se utiliza en la construcción de páginas web. Presenta elementos que proporcionan el diseño básico para un sitio web. Además de darles estructura incorpora un buen número de elementos visuales que pueden «decorarse» y procesarse. Sus mayores ventajas son la simplicidad, compatibilidad con múltiples navegadores y posibilidad de combinar con otros idiomas. Sus principales desventajas son la naturaleza estática, la seguridad y la gran cantidad de código necesaria para construir páginas relativamente sencillas.
- **CSS** (*Cascading Style Sheets*, hojas de estilo en cascada) es el lenguaje que se utiliza para modificar la estética de los elementos HTML. CSS incluye soporte para múltiples navegadores, es fácil de usar por su parecido con el inglés y relativamente rápido. Sin embargo, suelen producirse incompatibilidades entre navegadores y su aprendizaje suele ser costoso para los principiantes.
- **JavaScript** es el lenguaje de desarrollo del lado cliente más importante y popular. Proporciona flexibilidad y capacidad de respuesta rápida al sitio web. Sus ventajas son la simplicidad, la velocidad y la funcionalidad adicional o ampliada que proporciona al resto de elementos. Pero, al mismo tiempo, son programas difíciles de depurar y a veces se producen incompatibilidades entre navegadores. Es también el lenguaje utilizado por los desarrolladores *full stack*.

Además de estas tres tecnologías, existen otras que complementan, aceleran y optimizan el desarrollo de aplicaciones web del lado cliente. Aunque existe un inmenso catálogo disponible, se citan algunas de las más utilizadas:

- **SASS** (*Syntactically Awesome Style Sheets*, hojas de estilo sintácticamente asombrosas) es un preprocesador de CSS. Dicho de otra manera, es una tecnología con la que pueden generarse de forma automática hojas de estilo que contienen elementos propios de los lenguajes de programación, y que no tiene CSS, como pueden ser variables, funciones, etc. Su principal ventaja es la reutilización del código. Como desventaja se puede citar que debe compilarse y que la resolución de problemas no siempre es una tarea sencilla.

- **jQuery** es una librería de JavaScript que mejora el procesamiento del código HTML, el manejo de eventos y las animaciones. Es muy conciso y reduce drásticamente la cantidad de líneas del código.
- **vue.js** es un *framework* JavaScript creado por un ex ingeniero de Google para crear aplicaciones web compactas. Prácticamente eliminó los inconvenientes de los dos siguientes *frameworks* facilitando el desarrollo *front-end*.
- **AngularJS** es un *framework* JavaScript lanzado por Google que proporciona elementos más atractivos a las plantillas HTML y aumenta su rendimiento.
- **React** es otro *framework* JavaScript, lanzado por Facebook, muy popular, que mejora los componentes de la interfaz de usuario y provee de mucho más dinamismo a las aplicaciones web. Se utiliza principalmente en sitios web de alto tráfico y mucha interactividad. Es el preferido por los desarrolladores seguido de vue.js y luego Angular.

■ ■ ■ Actividad propuesta 1.2

Frameworks de JavaScript

Busca aquellos *frameworks* de desarrollo para JavaScript más usados por la comunidad de programadores e interioriza cuáles son los pros y los contras de cada uno de ellos, y para qué tipo de proyectos se recomiendan.

¿Qué combinación de tecnologías es la ideal? Depende enormemente del tipo de aplicación que se vaya a desarrollar. Una aplicación puede necesitar mucha interactividad asíncrona, otra mucha carga de procesamiento rápido y otra un nivel superior en la presentación de los datos. Cada uno de los *frameworks* y librerías anteriores, y otros, marcan la diferencia en un tipo de aplicación concreto. Sin embargo, a la vista de las tecnologías más utilizadas en la actualidad no cabe ninguna duda de que es necesario dominar en profundidad JavaScript, mucho más cuando se conoce que casi todas las tecnologías anteriores se han construido sobre JavaScript.

■ ■ ■ 1.2.3. ¿Por qué JavaScript?

Para conocer la respuesta a esta pregunta se enumeran antes algunos de los méritos de este lenguaje:

- Es muy fácil de implementar: tan solo es necesario colocar el código en un documento HTML y decirle al navegador que es JavaScript.
- Funciona en todos los navegadores que usan los usuarios de la web, incluso cuando están desconectados.
- Permite crear interfaces altamente amigables que mejoran la experiencia del usuario y brindan mucho dinamismo, sin tener que esperar a que el servidor reaccione y muestre otra página.
- Puede cargar contenido de forma asíncrona en el documento si el usuario lo necesita y cuando lo necesite, sin recargar toda la página.

- Puede comprobar lo que es posible hacer en un navegador y reaccionar en consecuencia.
- Puede ayudar a solucionar problemas de incompatibilidades entre navegadores y corregir problemas de diseño con CSS.

```

document.getElementById('div').innerHTML = '';
else if (i==2) {
    var atpos=inputs[i].indexOf('@');
    var dotpos=inputs[i].lastIndexOf('.');
    if (atpos<1 || dotpos<atpos+2 || dotpos>inputs[i].length-2)
        document.getElementById('errEmail').innerHTML = 'Email incorrecto';
    else
        document.getElementById('div').innerHTML = 'Email correcto';
}

```

Figura 1.4. Fragmento de código JavaScript, «el lenguaje de la web».

Prácticamente ninguna de las virtudes anteriores estaba disponible en las tecnologías que se utilizaban para aportar dinamismo a los sitios web, antes de la popularización de JavaScript. Esto hizo que poco a poco se comenzara a adoptar JavaScript en detrimento de otras opciones. Junto con su adopción, los navegadores fueron mejorando significativamente sus motores e incrementando su velocidad de procesamiento, lo que hizo que JavaScript empezara a mostrar todo su potencial. Tal ha sido su adopción, que en la actualidad alrededor del 95 % de todas las aplicaciones web se crean con JavaScript.

Otro de los aspectos que han conseguido posicionar a JavaScript como «el lenguaje de la web» ha sido su enorme capacidad de adaptación a los tiempos. Estamos ante un lenguaje que ha evolucionado notablemente, y que ha ido incorporando hasta nuestros días todas aquellas características de otros lenguajes y de la propia industria que le han permitido seguir siendo la opción más ventajosa para los programadores. En la actualidad, existen desarrollos de JavaScript que pretenden incorporar el acceso al portapapeles o el propio sistema de ficheros de los equipos cliente. Y estos son solo dos ejemplos de la tremenda implicación y capacidad de adaptación del lenguaje a cada momento histórico.

Teniendo en cuenta todo lo anterior, es importante señalar que JavaScript también tiene sus problemas en contextos muy concretos que, para algunos expertos, son insalvables. Por ello, existen algunas alternativas interesantes, como TypeScript, ELM, Dart o CoffeeScript, que no superan a JavaScript en una amplia generalidad de casos, pero sí que aportan detalles diferenciadores en escenarios muy específicos.

1.3. Herramientas para el desarrollo en el lado cliente

Antes de iniciar el recorrido en el aprendizaje de JavaScript es necesario tener bien establecido un entorno de trabajo.

Existen docenas de herramientas que correctamente configuradas pueden llevar a otro nivel el desarrollo de aplicaciones web en entorno cliente. Pero eso será más tarde, cuando sea necesario aumentar la productividad, gestionar cantidad de ficheros o trabajar en equipos de desarrollo. De momento hay que centrarse en lo imprescindible para aprender las bases del lenguaje: un navegador web, un editor de código y un intérprete de JavaScript.

1.3.1. Navegador

El navegador web es una de las principales herramientas, puesto que sin él no pueden verse las aplicaciones ejecutándose en su contexto. Atendiendo al porcentaje de uso de los diferentes navegadores, se ve claramente el nivel de adopción absolutamente dominante de Google Chrome (junio de 2022):

Chrome	Safari	Edge	Firefox	Samsung	Opera
65,87%	18,62%	4,12%	3,26%	2,88%	2,12%

Cuota de mercado de navegadores - Junio 2022

■ Chrome ■ Safari ■ Edge ■ Firefox ■ Samsung Internet ■ Opera

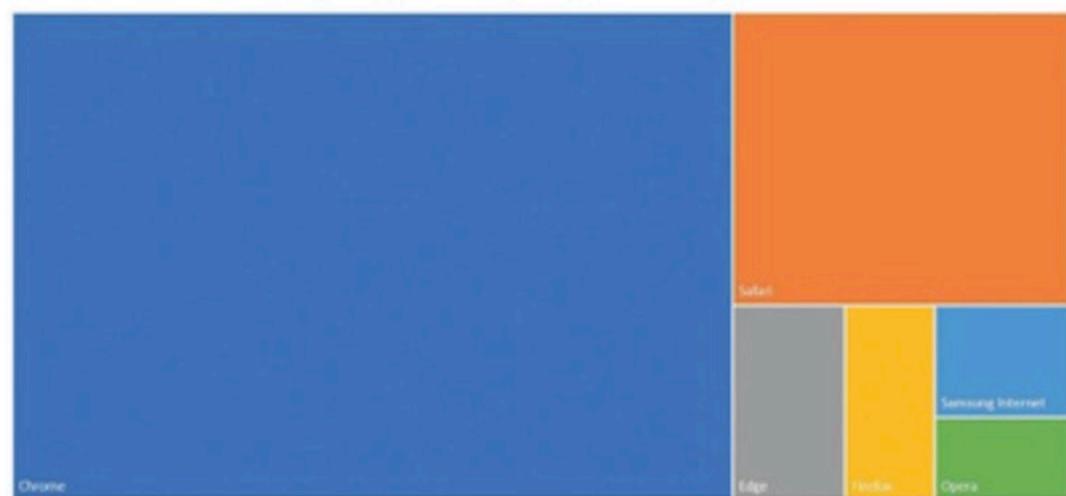


Figura 1.5. Diagrama por bloques con la proporción de adopción de cada navegador.
(Fuente: gs.statcounter.com).

¿Quiere decir esto que hay que desarrollar para Chrome? No, ni mucho menos. Todos los desarrolladores deben trabajar con varios navegadores. Por pura matemática, centrarse solo en el dominante deja fuera de las optimizaciones a casi un 35 % de los usuarios, y eso es mucha cuota de mercado.

De la misma manera, optimizar las aplicaciones para todos los navegadores, además de ser técnicamente muy complicado, tiene poco sentido: no compensa la cantidad de esfuerzo y recursos necesarios invertidos en relación con el número de usuarios que se alcanza.

Otra cuestión interesante que no debe perderse de vista es que estos datos son globales. En ocasiones se realizan proyectos para áreas geográficas o sociales en concreto (comunidades, países u organizaciones) que usan un conjunto muy reducido de navegadores, en ocasiones solo un par de ellos, o con una dominancia distinta, por lo que es completamente superfluo invertir recursos en una alta compatibilidad concreta entre ellos.

Por todo ello, lo que se recomienda es probar y ajustar las aplicaciones para los cuatro navegadores con mayor cuota de mercado entre la audiencia objetivo de la aplicación, optimizándola todo lo posible para el navegador dominante.

Además, también es fundamental probar diferentes configuraciones del navegador para ajustarlo al mayor número de tamaños de pantalla posibles. En esto, puede ayudar mucho la oferta de *plugins* o extensiones de los navegadores. En prácticamente todos ellos, pueden instalarse simuladores de dispositivos que muestran nuestras aplicaciones ejecutándose en cualquier tamaño de pantalla disponible en el mercado.

1.3.2. Editor de código

Bien, tenemos un navegador donde ver el resultado de nuestro trabajo, pero nuestro trabajo debe escribirse en algún sitio utilizando alguna herramienta. Los editores de código permiten a los programadores abrir múltiples archivos de código ayudándoles a acortar el tiempo de desarrollo y a realizar las tareas de forma más eficiente.

Así como un procesador de texto enriquecido es una aplicación personalizada para editar documentos, con muchas funciones avanzadas para escribir y editar, un editor de código tiene muchas funciones personalizadas especialmente diseñadas para satisfacer las necesidades de un desarrollador de software. Algunas de estas características incluyen:

- **Comprobación de errores** conforme se va escribiendo, lo que proporciona un primer nivel de corrección.
- **Sugerencias de autocompletado**, que agiliza la escritura y acorta los tiempos de desarrollo.
- **Fragmentos de código** previamente configurados con aquellas porciones utilizadas con mucha frecuencia y que evitan reescribirlos constantemente.
- **Resaltado de sintaxis**, que ayuda mucho a identificar cada uno de los elementos de los programas con un simple golpe de vista.
- **Navegación entre archivos y recursos**, que ayuda a tener localizados todos los ficheros en sus ubicaciones correctas.
- **Funcionalidad extendida** a través de complementos específicos de las tecnologías con las que se trabaja en un proyecto en concreto.

A menudo se escuchan conceptos como «editor de código» y «entorno de desarrollo integrado» usados como sinónimos, cuando en realidad se trata de dos herramientas muy distintas.

Si como se ha visto, un editor de código sirve principalmente para crear un código más limpio y eficiente gracias a las características y funciones que incorpora, un IDE (entorno de desarrollo integrado) es un conjunto de herramientas combinadas. Los entornos de desarrollo integrados toman su nombre de la integración de otras herramientas avanzadas, como depuradores, analizadores de código, optimizadores, compiladores, controladores de versiones y una gran cantidad de otras características que ayudan a un desarrollador a moverse a lo largo del ciclo de vida del desarrollo del software.

Entonces, ¿es necesario un editor o un IDE? Como casi todas las preguntas en el contexto del desarrollo de aplicaciones web, todo depende del tipo y tamaño de la aplicación que vaya a construirse. Y no solo de eso, sino de la propia tarea que sea preciso completar en un momento dado. Sin ninguna duda, si estamos en un equipo de desarrollo trabajando en un proyecto grande con múltiples áreas de trabajo, vamos a necesitar un IDE. Por el contrario, si nuestro proyecto es pequeño, identificamos y corregimos con rapidez los errores y no necesitamos ninguna de las herramientas integradas de un IDE, ¿por qué gastar recursos en herramientas que no vamos a usar? Por último, si estamos resolviendo un error puntual de forma rápida, podemos incluso prescindir de los dos tipos y utilizar algún editor *online* que permite probar al vuelo fragmentos de código HTML, CSS y JavaScript sin necesidad de instalar un entorno de desarrollo.

Centrándonos en las características de los editores de código que pueden marcar la diferencia, estos podrían ser los criterios más importantes en los que deberíamos fijarnos a la hora de elegir un editor u otro:

- **Funciones básicas imprescindibles:** resaltado de sintaxis, sangría automática, finalización automática, coincidencia de bloques y visualización de números de línea.
- **Experiencia del usuario:** reducir todo lo posible las dificultades para escribir código y aumentar así la eficiencia en la producción de código final.
- **Facilidad de aprendizaje:** invertir tiempo en entender el editor cuando aún no se domina un lenguaje es una auténtica pérdida de tiempo, por lo que sería deseable descartar de inicio aquellos editores menos intuitivos.
- **Extensibilidad:** que proporcione vías para extender las capacidades del editor con más funciones y herramientas adicionales.
- **Velocidad:** la rapidez es clave a la hora de codificar y no todos los editores responden bien cuando se realizan ciertas tareas en equipos cortos de hardware.
- **Sistema operativo:** mucho mejor si el editor provee versiones para al menos Windows, Linux y MacOS.
- **Compatibilidad con Git:** dado que Git se ha convertido en un estándar de facto como sistema de control de versiones, sería deseable que incorporara esta funcionalidad de forma nativa o como extensión.
- **Soporte de la comunidad:** tener un lugar muy frecuentado para hacer preguntas y obtener soporte técnico rápido es crucial para la mayoría de los desarrolladores.

Un error del editor en el peor momento y sin capacidad de soporte puede marcar la diferencia entre un proyecto exitoso y otro fallido.

- **Compatibilidad con lenguajes de programación:** es prácticamente seguro que será necesario escribir ficheros con distintos lenguajes en un mismo proyecto. Por lo que es imprescindible elegir un editor que proporcione todas las herramientas imprescindibles para todos ellos.
- **Precio:** existen muchos editores de altísima calidad que son completamente gratuitos, otros aportan funcionalidades *premium* y un pequeño número de ellos tienen planes de pago. Siempre es recomendable analizar qué ofrecen porque en ocasiones una pequeña inversión puede resultar en una sabia decisión que marque la diferencia.
- **Atajos de teclado:** los métodos abreviados de teclado ayudan mucho a los desarrolladores a crear fragmentos de código más rápido sin necesidad de utilizar el ratón constantemente.
- **Ventana de vista previa:** otra característica interesante que puede ahorrar mucho tiempo evitando cambiar entre editor y navegador cuando se realizan pequeños cambios.
- **Gestión amigable de errores:** los editores que incorporan un sistema de coloreado e información ampliada de los errores facilitan de forma muy significativa la localización de errores y los problemas del código.

Como recomendación de algunos editores muy populares pueden incluirse los siguientes:

- **Editores de código:** Atom, Sublime Text, Brackets, Coda...
- **Entornos de desarrollo integrados (IDE):** Visual Studio, Eclipse, Netbeans, Webstorm...
- **Editores online:** CodePen, JSFiddle, JS Bin...

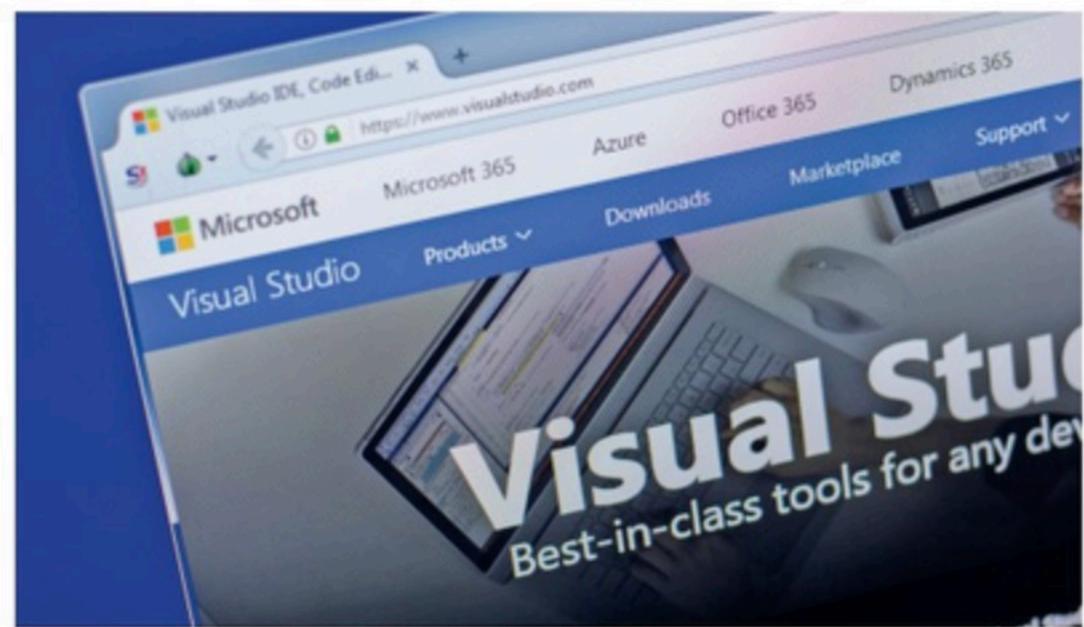


Figura 1.6. Visual Studio Code es el IDE más utilizado por la comunidad de programadores.

Navegando por internet se encuentran cientos de artículos y guías sobre cuáles son los mejores editores de código y cuáles se deberían usar. Sin embargo, la mejor recomendación que puede realizarse a cualquier desarrollador de software es que pruebe todos los que pueda. Vamos a estar miles de horas utilizando un editor, por lo que es imprescindible sentirse cómodos con él. A veces los programadores prefieren utilizar dos herramientas, o tres, o las que sean, porque son con las que se encuentran cómodos, y con las que son más productivos. La comodidad de cada persona es algo completamente subjetivo, por lo que es preciso probar el mayor número de herramientas posibles, para encontrar el entorno que proporcione el mayor confort. Esa es, en muchas ocasiones, la clave de la productividad de un programador.

Actividad propuesta 1.3

Editores de código

Navega hasta las páginas oficiales de Sublime Text y Visual Studio Code e instala ambas aplicaciones. Crea una página HTML simple con un pequeño CSS asociado y comienza a familiarizarte con ambos entornos.

1.3.3. Intérprete de JavaScript

Evidentemente para escribir código JavaScript es necesario tener el propio intérprete del lenguaje, que permitirá probar los programas para saber si lo que se ha escrito es correcto o no.

Por la propia naturaleza de esta tecnología no es preciso realizar ninguna instalación adicional si se dispone de un navegador web, puesto que JavaScript viene integrado en ellos. Lo que sí debe comprobarse es que esté habilitado, porque en muchas ocasiones los usuarios, consciente o inconscientemente, lo tienen deshabilitado. A continuación, se explica cómo habilitarlo en los cuatro principales navegadores.

- **Chrome:** entrar en la **Configuración**, seleccionar **Seguridad y privacidad**, clicar en **Configuración de sitios** y después en **JavaScript** y finalmente seleccionar **Los sitios pueden usar JavaScript**.
- **Safari:** acceder a **Herramientas**, clicar en **Preferencias**, acceder a **Seguridad** y activar la casilla etiquetada como **Activar JavaScript**.
- **Edge:** acceder al menú de **Opciones** y desplegar **Mostrar opciones avanzadas**. En la sección de **Privacidad** clicar en **Configuración de contenido**. Navegar hasta la sección **JavaScript** y finalmente clicar en **Permitir que todos los sitios ejecuten JavaScript (recomendado)**.
- **Firefox:** hacer clic en **Herramientas**, luego en **Opciones** y en la pestaña **Contenido** hacer clic en la casilla **Activar JavaScript**.

En todos los casos, debe reiniciarse el navegador para tener un intérprete que es capaz de ejecutar los programas JavaScript.

1.3.4. Otras herramientas útiles

Como ya se indicado anteriormente, puede resultar interesante la utilización de otras herramientas que, si bien no son imprescindibles en una generalidad de casos, podrían ser completamente necesarias.

Node.js

Como su propia documentación indica, Node.js es una plataforma construida sobre el motor V8 de Google Chrome para el desarrollo fácil, rápido y escalable de aplicaciones web. Está especialmente indicado para aplicaciones en tiempo real con un consumo intenso de datos que corren sobre dispositivos distribuidos.



Figura 1.7. Node.js es el siguiente paso natural tras aprender las bases de JavaScript.

La gran potencia de esta herramienta radica en que convierte a nuestra máquina en un servidor web y es capaz de crear código JavaScript del lado servidor. Sí, no es ningún error, utiliza JavaScript para el lado cliente y para el lado servidor.

Como ya se podrá intuir estamos ante una de las herramientas más utilizadas para el desarrollo de proyectos de cierto tamaño. Es muy importante conocer su funcionamiento y sus capacidades para desarrolladores *full stack*. Sin embargo, y siguiendo nuestro criterio de no usar aquello que no es imprescindible, esta tecnología queda fuera del alcance de una obra dedicada al lado cliente. No obstante, Node.js también interpreta código JavaScript del lado cliente.

Si el lector quisiera probar los conceptos de JavaScript sin necesidad de crear una página web o sin necesidad de utilizar un navegador, esta sería la opción recomendada.

Git

Git es el sistema de control de versiones más utilizado. En un lenguaje llano, Git es útil para cualquier profesional que escriba código o necesite hacer un seguimiento de los cambios que se han producido en los archivos.

Además, también facilita la colaboración, permitiendo que los cambios realizados por varias personas se fusionen en una sola fuente.

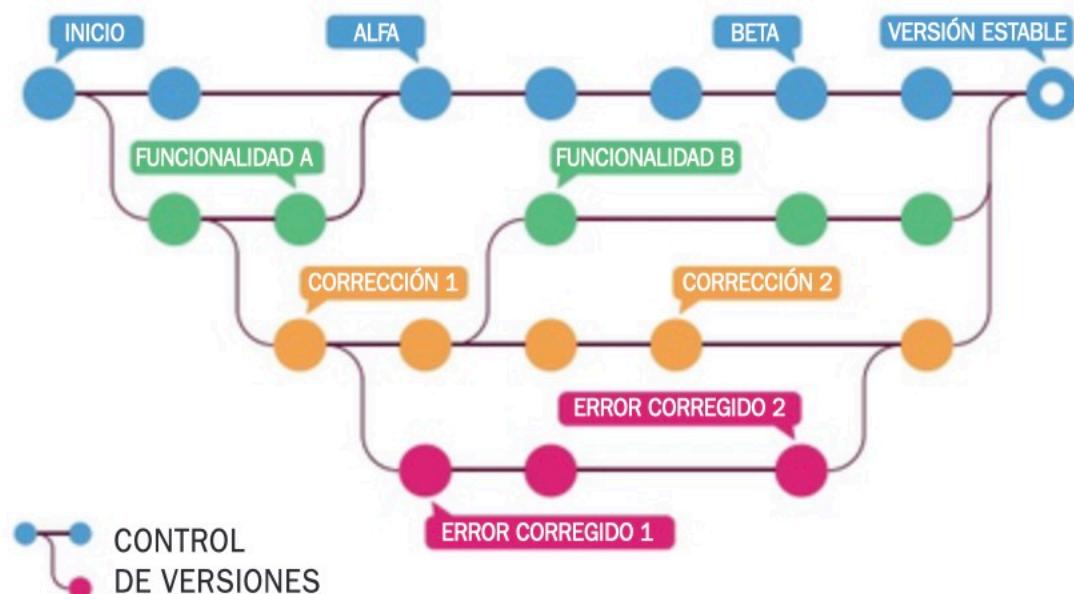


Figura 1.8. Ejemplo gráfico de las ramificaciones y fusiones de ramas en un proyecto con Git.

Git es un software que se ejecuta localmente, por lo que los archivos y su historial de cambios de almacenan en las máquinas clientes. También es posible usar equipos *online*, como GitHub, para almacenar una copia de los archivos y su historial de revisiones. Tener un lugar centralizado permite cargar los cambios y descargar los cambios de otros, para colaborar de una manera eficaz en aquella versión más prometedora. Git permite fusionar automáticamente los cambios, por lo que dos personas pueden incluso trabajar en diferentes partes del mismo fichero y luego fusionar ambos sin perder el trabajo de los demás.

Para saber más

En estos dos enlaces o con los códigos QR puedes encontrar guías del autor con información muy útil para empezar a trabajar con estas dos herramientas: Node.js y Git.

<https://lopegonzalez.es/nodejs>

Sería muy deseable que cuando adquieras cierta destreza escribiendo y ejecutando las piezas de código que aparecen en esta obra, vuelvas atrás, installes NodeJS y realices más pruebas utilizando este entorno de ejecución.



<https://lopegonzalez.es/git>

Igualmente, puedes «entrenar» tus habilidades con GIT, creando dos usuarios y modificando al mismo tiempo una pieza de código para poder experimentar cómo se trabajaría con un equipo de programadores.

