1. INTRODUCCIÓN A LA PROGRAMACIÓN WEB

La **programación web** es el conjunto de técnicas y tecnologías que permiten desarrollar sitios web y aplicaciones accesibles a través de navegadores web. Los **sitios web** son archivos que se descargan en nuestros navegadores desde ordenadores remotos. Cuando accedemos a un sitio web, informamos al navegador de la dirección del sitio y el programa descarga los archivos, procesa su contenido y lo muestra pantalla. Estos **sitios web** son de acceso **público** e internet es una red global, por lo que estos archivos están siempre disponibles, así que los sitios web se almacenan en ordenadores especializados diseñados para despachar archivos a los usuarios que lo solicitan. El ordenador que almacena estos archivos y datos de un sitio web se llama **servidor** y el ordenador que accede a esta información se llama **cliente**.

Los servidores son ordenadores similares a los personales, pero con la excepción de que siempre están conectados a la red y ejecutando programas que permiten responder a las solicitudes de los usuarios. Uno de los servidores web más populares es Apache. Este programa permite establecer la conexión entre el cliente y el servidor, controlar el acceso de los usuarios, administrar los archivos y despachar los documentos y recursos requeridos por los clientes.

En los **sitios web** nos encontramos **muchos documentos** que el navegador **descarga** cuando el usuario los **solicita**. Los **documentos** que conforman un sitio web se llaman **páginas** y el proceso de abrir nuevas páginas **navegar**. Para desarrollar un sitio web, tenemos que crear un archivo por cada página que queremos incluir. Junto con estos archivos, también podemos incluir archivos con imágenes y otros recursos que queramos mostrar en las páginas.

Normalmente, el **archivo index.html** contiene el código y la información correspondiente a la página principal. **Cuando** un usuario **accede** a nuestro **sitio web por primera vez**, el navegador **descarga el archivo** index.html **y muestra su contenido en la ventana**. El **resto** de las páginas creadas **se** irán **cargando conforme** el usuario las vaya **solicitando**.

Los **archivos** de un sitio web son iguales que los archivos que podemos encontrar en un ordenador personal. **Todos tienen un nombre seleccionado por el desarrollador y una extensión que refleja el lenguaje usado para programar su contenido.** Aunque podemos asignar cualquier nombre que queramos a estos archivos, el archivo que genera la página inicial presenta algunos requisitos. Algunos servidores como Apache designan archivos por defecto, si el usuario no especifica ninguno, el nombre que buscará el navegador será índex, será el punto de entrada a nuestro sitio web y por eso debemos **añadirlo a nuestro proyecto**. Aunque índex es el nombre más común, algunos servidores pueden

designar otros como home o default, y podemos encontrar distintas extensiones: html, php o js. Así Apache primero busca un archivo index.html, si no lo encuentra busca index.php, veremos las distintas opciones que nos encontramos.

La **web** se basa en el concepto de **hipertexto**, una t**ecnología que permite enlazar diferentes documentos y recursos a través de enlaces o hipervínculos**. Esta capacidad de enlazado es lo que permite la navegación entre diferentes páginas y sitios web. **El World Wide Web Consortium (W3C) es la organización que se encarga de desarrollar estándares para la Web**. Su objetivo es garantizar que la Web continúe siendo abierta, accesible y funcional para todos. Estos estándares incluyen tecnologías fundamentales como HTML, CSS y JavaScript.. La web se compone de varios elementos clave:

- **Hipertexto**: Permite la creación de **enlaces** entre documentos.
- **Protocolo HTTP:** Es el protocolo de comunicación que permite la **transferencia** de **datos** en la web.
- **Servidores y clientes**: Los servidores **alojan** sitios web y los clientes (navegadores web) los **solicitan** y los **muestran**.
- **DNS (Sistema de Nombres de Dominio): Traduce** los nombres de **dominio** legibles por humanos (como www.ejemplo.com) en direcciones IP que los ordenadores lo puedan entender.
    - En la URL nos podemos encontrar tres partes:

        - **Protocolo de comunicación** que utilizamos para acceder al recurso. Existen diversos protocolos para transferir recursos y datos como:
            - HTTP: Protocolo que se utiliza para acceder a documentos web, aunque no es necesario escribirlo, ya que los navegadores lo hacen de forma automático.
            - HTTPs: Conexión encriptada por protocolos de encriptación como TLS o SSL.
        - **Dominio del sitio web.**
        - **Nombre del recurso a descargar** (puede ser un archivo o una ruta para seguir donde incluye el directorio donde está el archivo almacenado).
    - También es necesario diferenciar entre URL absolutas y relativas:

        - **URL absolutas:** Son aquellas que incluyen toda la información necesaria para acceder al recurso.
            - Ejemplo**:** https://www.ejemplo.com/contacto
        - **URL relativas:** Solo declaran la parte de la ruta que el navegador que el navegador tiene que agregar a la URL
            - Ejemplo**:** /contacto
    - Una vez que el sitio web está listo para ser presentado en público, tenemos que registrar el dominio que los usuarios van a a escribir en la barra de navegación para acceder a él. El nombre de nuestro dominio puede cualquiera, y contamos con varias opciones para definir la extensión, desde extensiones con propósitos comerciales, a aquellas sin ánimo de lucro y personales, como .org, .net o .info, por no mencionar las extensiones regionales que incluyen un valor adicional para determinar la ubicación del sitio web. Para obtener un dominio para nuestro sitio web, tenemos que abrir una cuenta con un registrante y adquirirlo. Los dominios requieren del pago de un arancel anual, pero el proceso es relativamente sencillo y hay muchas compañías disponibles que pueden hacerse cargo del trámite por nosotros. Lo más importante es asegurarnos que el nombre elegido no está siendo utilizado y está disponible, y luego hacer el pedido.

Como ya nombraremos con mayor detalle más adelante, el desarrollo de aplicaciones web es un campo multidisciplinario que abarca varias áreas y tecnologías como son:

- **Contenido**: Se refiere a la organización, semántica, presentación y estructura de la información en la web.
- **Diseño**: El diseño se centra en el aspecto visual y los elementos gráficos de una página web.
- **Tecnología Frontend:** Son las tecnologías que se ejecutan en el navegador del usuario y son responsables de los elementos interactivos de una página web.
- **Distribución**: Se refiere a como se **despliegan y gestionan los sitios web**. Esto incluye tanto el hardware (servidores, máquinas virtuales, contenedores) como el software (servidores web Apache, IIS y NGINX)
- **Propósito**: Las web se hacen con un **objetivo** para trasladar a las necesidades de los clientes a los desarrolladores o inventar nuevos modelos de negocio.

Por lo tanto, en este módulo **no** nos vamos a centrar demasiado en la parte **estética** (aunque es inevitable construir HTML y reaccionar a las acciones del usuario), vamos a suponer que hay un

servidor funcionando correctamente y nos vamos a centrar en **cómo recoger dos datos del servidor, como mostrarlos y hacer algo con ellos.**

1. Lado cliente y lado servidor:

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

- **Lado del 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 ejecuten en el navegador**. Desarrollan los diseños de la interfaz de usuario y su experiencia, que son los elementos clave que dan vida a la interfaz.
    - Clientes web: Los navegadores como Firefox, Chrome, Vivaldi, Opera, Edge e Internet Explorer.
    - Lenguaje de marcas: HTML, XHTML, HTML5 y CSSS.
    - Lenguajes de programación del entorno cliente: principalmente JavaScript.
- **Lado del 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 está programada la lógica de negocio de la aplicación.**
    - Hardware: Incluye servidores y elementos de red, máquinas virtuales y contenedores.
    - Software: Involucra servidores web (Como Apache, IIS, NGINX) y lenguajes CGI (Como Perl, PHP, C). También incluye lenguajes y framework con servidores web integrados, como Python, Java, node.js y C++.

En el desarrollo web, tanto el lado del servidor como el lado del cliente juegan sus propios **roles**. Sin embargo, muchas **tareas** pueden llevarse a cabo en cualquiera de los **dos lados**, y la elección de donde implementar una funcionalidad específica depende de varios factores, como la tecnología adecuada para el caso o la comodidad del desarrollador de una tecnología particular. Las tareas más comunes en ambos lados son:

- **Validar formularios:** La validación se puede hacer desde ambos lados, pero se debe tener en cuenta las siguientes preferencias:

    - Desde el lado cliente: La validación se puede hacer en el navegador del usuario para que el usuario tenga una experiencia más rápida y fluida. Por ejemplo, una comprobación de que un campo del formulario no está vacía, para evitar así que el usuario envié datos incorrectos.
    - Desde el lado servidor: La validación en este lado es por razones de seguridad, y es necesario volver a comprobar los datos en el servidor antes de procesarlo, ya que un "hacker" podría eludir esta validación.
- **Guardar datos permanentes:** Generalmente se hace en el servidor utilizando bases de datos. Pero también se puede almacenar información en el navegador del usuario como Local Storage que permite guardar preferencias de tema de un usuario o configuraciones, como podría ser el carrito de compras que tenía creado después de cerrar la página o las cookies… Es decir, en el lado del cliente solo se almacena información para mejorar la experiencia del usuario y al contrario, en el servidor para almacenar datos con mayor seguridad y asegurar la integridad.
- **Presentación de diapositivas (imágenes, videos, gráficos..)**: Es más común en el lado del cliente utilizando JavaScript o framework de frontend. Se podría hacer desde el lado del servidor, pero habría que realizar una nueva solicitud al servidor y sería mucho más lento y menos interactivo.
- **Modificar el contenido visible de una pagina**, como eliminar un párrafo, se usa JavaScript. Este proceso se hace directamente en el DOM (Document Object Model), lo que permite cambios instantáneos en la interfaz del usuario sin necesidad de comunicarse con el servidor.
- **Calcular datos:** Los cálculos pueden hacerse en ambos lados, dependiendo de la naturaleza del cálculo y los requisitos de seguridad.
    - En el lado del cliente: Tareas rápidas, como por ejemplo podría ser sumar números en un formulario.
    - En el lado del servidor: Tareas con datos sensibles o críticos, como podría ser cálculos financieros, es mucho más seguro realizarlos en el servidor.
- **Acceder a bases de datos:** Se realiza en el servidor para mantener la seguridad y la integridad de los datos. Aunque hay algunas situaciones en las que el cliente puede interactuar de forma indirecta, por ejemplo con una API RESTful el cliente puede hacer solicitudes a una API para obtener datos o enviar datos, en este caso no se accede de forma directa, ya que interactúa con el servidor, pero obtiene los datos.

Las principales diferencias entre el desarrollo de los lados cliente y servidor:

	Lado cliente	Lado servidor
Definición	Implica la **implementación efectiva** de los **componentes visuales** en una aplicación web	Implica la **implementación efectiva** de **funcionalidades** de una **aplicación web,** donde se incluyen en el acceso a datos, la administración de servidores, etc..
Habilidades requeridas	HTML, CSS, SASS, JS…	Python, PHP, Java, Ruby…
Independencia	**No** puede funcionar de forma **independiente** excepto en el caso de sitios estáticos.	Funciona de forma **independiente** respecto del front-end.
Objetivo	Garantizar que todos los usuarios puedan **acceder** a la aplicación y que siga **respondiendo** en todos los dispositivos.	Garantizar que la aplicación se **ejecuta** 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 del usuario.	Su trabajo es proporcionar datos al front-end, vincular páginas, brindar seguridad y soporte a los usuarios.
Frameworks utilizados	AngularJS, React, vue.js…	Django, Laravel, Flask…
Habilidades adicionales	Una buena comprensión del diseño de UI y UX	Razonamiento lógico y resolución de problemas.

Se llama **framework** al conjunto de **herramientas** y librerías **software** predefinidas que proporcionan una **estructura base** y numerosas utilidades para el desarrollo de aplicaciones web más escalables y sencillas de mantener, por lo que conducen a un importante ahorro de recursos. Las ventajas que proporciona son:

- Proporcionan una arquitectura estándar que facilita la escalabilidad y el mantenimiento de grandes aplicaciones.
- Automatiza tareas comunes para que el desarrollador no tenga que reinventar nada.
- Reduce tiempos de desarrollo porque muchas funciones vienen ya implementadas.
- Ayuda a mantener el código organizado.

Una **biblioteca** es un conjunto de **funciones y métodos** que se pueden utilizar cuando se necesite, pero que no impone una estructura. Se puede decidir cuándo y cómo utilizarlo dentro del código. Algunas de sus ventajas son:

- Mayor flexibilidad, no obligándote a seguir una arquitectura o patrón específico. El desarrollador es quien controla el flujo del programa.
- Ideal para solucionar tareas específicas: manejar fechas, hacer peticiones HTTP..
- Se pueden utilizar varias bibliotecas en una misma aplicación sin problema.

Para obtener las bibliotecas o framework y poder utilizarlas en nuestros futuros proyectos:

- Descarga directa de un archivo .js y copiarlo al proyecto.
- Utilizar un CDN directamente desde la web oficial. (No se descarga localmente, el sitio web lo descarga directamente desde internet
- Utilizar herramientas como NPM o Yarn para gestionar dependencias.

Por otra parte**, UX (User eXperience) y UI (User Interface)** son conceptos muy importantes en el desarrollo de aplicaciones web:

- **UX:** Determinan la satisfacción de uso de una aplicación. Se centra en el diseño de la **experiencia global** del usuario, lo cual implica asegurar que la aplicación sea fácil de usar, eficiente, y genere una interacción satisfactoria. Objetivos: Satisfacción del usuario, intuitiva, resolución de problemas, eficiencia...
- **UI:** Se enfoca en la **interacción visual** y los **elementos gráficos** que permiten al usuario interactuar con la aplicación o sitio web. Trata sobre **cómo se ve** y **cómo se siente** la aplicación visualmente. Objetivos:

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 desarrollares que son capaces de crear tanto la parte frontal como la parte trasera de una aplicación web, fundamentalmente con tecnologías construidas sobre JavaScript.

2. Arquitectura:

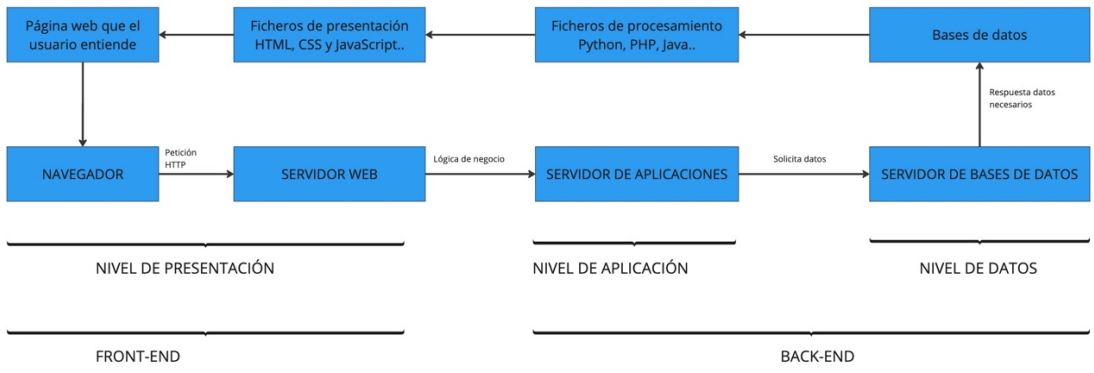
Dejando a un lado diferencias sobre cada lado de una aplicación, es también muy importante conocer la **arquitectura** de una **aplicación web típica.**

Las aplicaciones web de hoy en día utilizan las **tecnologías ambos lados** siguiendo una **arquitectura de tres niveles.** Se trata de la arquitectura software **predominante** para las aplicaciones **cliente-servidor** tradicionales. Organiza las aplicaciones en tres niveles informáticos, lógicos y físicos:

- **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 (botones, menús, formularios, etc.). 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,** el cual proporciona estas páginas al navegador del cliente.
- **Nivel de aplicación:** También conocido como **nivel lógico** o nivel 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 (validación de usuarios, envió de correos, etc.)** . Desde este nivel también se pueden **añadir, modificar o eliminar datos del nivel de datos ya que se comunica con la base de datos para almacenar o recuperar información.** 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, donde se ejecuta el código encargado de procesar la información.**
- **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 bases de datos relacional como MySQL, PostgreSQL, Oracle... o un servidor de bases de datos NoSQL como MongoDB, Cassandra… En el desarrollo web este nivel estaría representado como el s**ervidor de bases de datos ya que responde a las consultas hechas desde el nivel de 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.**

La lógica que sigue esta arquitectura es la siguiente:



1. El usuario a través del navegador realiza una acción. *(Ejemplo: Como puede ser introducir en la barra de búsqueda de una tienda online y pulsar el botón para buscar)*
2. Esta acción desencadena en una petición HTTP al servidor web. La petición puede ser de tipo GET (Para solicitar datos) o POST (para enviar datos), entre otras. *(Ejemplo: El navegador envía una solitud GET al servidor web solicitando los productos que coincidan con el término que se ha ingresado en la barra de búsqueda)*
3. El servidor web necesita realizar cálculos de procesamiento pertenecientes a la lógica de negocio (reglas y código que determinan cómo funciona la aplicación), por lo que solicita al servidor de aplicaciones que realice esta tarea. El servidor web es donde se aloja el sitio web, pero actúa como enlace entre el navegador del usuario con la petición y el servidor de aplicaciones con la lógica de negocio. *(Ejemplo: El servidor web pasa la solicitud al servidor de aplicaciones y este usa la lógica de negocio para buscar en la base de datos productos que coincidan con el término de búsqueda)*
4. Es posible que el servidor de aplicaciones necesite datos que están almacenados en una base de datos, por lo que solicita estos datos al nivel de datos. *(Ejemplo: El servidor de aplicaciones realiza una consulta a la base de datos para obtener una lista de los productos buscados y la base de datos responde con la lista de esos productos.)*
5. Una vez contemplado 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. *(Ejemplo: El usuario ve en su navegador una lista de productos que coinciden con el término de búsqueda que ingresó previamente. Ahora se puede seguir interactuando con la página)*

3. Desarrollo en el lado cliente:

El desarrollo de aplicaciones web ha evolucionado significativamente con el tiempo, y hoy en día existen diversas maneras de abordar el frontend.

3.1. Tecnologías del lado del cliente:

Como ya se ha nombrado con anterioridad, el **marco de ejecución de las aplicaciones web en el lado del cliente es el navegador.** Por este motivo es necesario dominar estas tres tecnologías:

- **HTML**: Es un **lenguaje de marcas** 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 necesario para construir páginas relativamente sencillas.

- **CSS**: Es el **lenguaje** de estilo 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 productos 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. Es también el lenguaje utilizado por los desarrolladores full Stack.
    - Algunos de los **méritos** de este lenguaje son:
        - 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** que mejoran la experiencia del usuario y brindan mucho **dinamismo**, sin tener que esperar a que el servidor reaccione y muestre otra página.
        - Puede ayudar a **solucionar problemas de incompatibilidades** entre navegadores y corregir problemas de diseño con CSS.
    - **Limitaciones** de JavaScript:
        - No puede escribir directamente en el servidor.
        - No todos los navegadores soportan JavaScript y se puede desactivar, por lo que una página web bien diseñada debería funcionar sin él.
        - No puede modificar preferencias del navegador, lanzar aplicaciones, leer o escribir archivos en el cliente, retrasmitir streaming, enviar emails, interactuar con lenguajes de servidor, acceder a una web de un dominio diferente, proteger origen de las imágenes, ni implementar multiprocesos.
    - **Tecnología asíncrona:** JavaScript puede solicitar datos al servidor de forma asíncrona utilizando tecnologías como AJAX.

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

- Con **JavaScript "Vainilla":** Utilización de JavaScript puro sin ninguna biblioteca o framework adicional. Más destinado para aplicaciones pequeñas y sencillas.
- Con bibliotecas:
    - **JQuery:** Es una librería de JavaScript que mejora el procesamiento el código HTML, el manejo de elementos y las animaciones. Es muy conciso y reduce drásticamente la cantidad de líneas del código. **Facilita la manipulación del DOM, eventos y AJAX.**
    - **Bootstrap**: Biblioteca de CSS y JavaScript para diseño responsivo (adaptables a dispositivos móviles).
    - **D3.js:** Para la visualización interactiva de datos en el navegador.
    - **Vue.js:** Ofrece una manera progresiva de construir interfaces de usuario.
    - **React**: Biblioteca para construir interfaces de usuario, principalmente en aplicaciones de una sola página (SPA).
- Con frameworks:
    - **Angular.JS**: Es un framework de JavaScript lanzado por Google que proporciona elementos más atractivos a las plantillas HTML y aumenta su rendimiento.
    - **React**: Aunque es una biblioteca, también se utiliza como framework cuando se combina con otras herramientas se considera un framework completo. Es otro framework JavaScript lanzado por Facebook, es 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 desarrollares seguido de vue.js y luego Angular.
    - **Vue.js**: Al igual que React, puede considerarse un framework cuando se utiliza en proyectos más grandes.
    - **Phaser**: Framework para el desarrollo de juegos en HTML5.
- Con frameworks Full Stack:
    - **Next.js**: Framework de React para aplicaciones de servidor y cliente.
    - **SvelteKit**: Framework para construir aplicaciones rápidas y modernas.
- Procesadores de CSS:
    - **SASS**: (Hojas de estilo sintácticamente asombrosas): Es una tecnología con la que pueden generarse de forma automática hojas de estilo que contienen elementos propias de los lenguajes de programación, y que no tiene CSS, como puede 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.

3.2. Herramientas para el desarrollo en el lado cliente:

Existen docenas de herramientas que correctamente configuradas pueden llevar a otro nivel el desarrollo de aplicaciones web en el 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 nos vamos a centrar en lo imprescindible para aprender las bases del lenguaje:

- **Navegador**: Es una de las principales herramientas, puesto que sin el no pueden verse las aplicaciones ejecutándose.
- **Editor de código**: Hasta hora, tenemos un navegador donde ver el resultado del trabajo, pero el trabajo debe escribirse en algún sitio utilizando alguna herramienta. Los editores de texto permiten a los programadores abrir múltiples archivos de código ayudándoosles a acorar el tiempo de desarrollo y realizar las tareas de forma más eficiente.
    - Software para modificaciones casuales ligero y rápido: Nano, Notepad..
    - Editores de texto avanzados: Sublime Text, Visual Studio Code, Vim..
    - IDEs: Netbeans, Eclipse…

El entorno de desarrollo que utilizaremos será Visual Studio Code y vamos a descargar las siguientes extensiones:

- **Prettier**: Para formatear archivos de manera automática, ya sea manualmente o al guardar.
- **EsLint**: Ayuda a detectar errores de estilo o programación que JavaScript no puede detectar por sí mismo (requiere instalar eslint en el proyecto).
- **Live Server**: Para probar la web en un servidor local de manera rápida y sencilla.

- **Intérprete de JavaScript:** Evidentemente para escribir 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 si se dispone de un navegador web, ya que suele venir integrado pero en muchas ocasiones se encuentra inhabilitado, por ello es necesario comprobar (Configuración → Seguridad y privacidad → JavaScript)
- **Otras herramientas útiles:**
    - Node.Js: Es una plataforma construida por el motor VS de Google Chrome para el desarrollo fácil, rápido y escalable de aplicaciones web. Esta especialmente indicado

para el desarrollo de aplicaciones en tiempo real con un consumo intenso de datos que corren sobre dispositivos distribuidos.

- La gran potencia de esta herramienta radica en que convierte a nuestra maquina en un servidor web y es capaz de crear código JavaScript del lado servidor, Si, utiliza JavaScript para el lado cliente y para el lado servidor.

- **Git**: Es un sistema de control de versiones más utilizado. Es 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.
  - Git es un software que se ejecuta localmente, por lo que los archivos y sus historias de cambios se almacenan en las maquinas 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.

4. Evolución de las aplicaciones web:

Las primeras aplicaciones web eran muy simples, se asemejaban a una carta escrita a papel ya que eran estáticas y sin interacción. Cada documento HTML contenía información que se podía leer, pero poco más. Las páginas contenían hipervínculos que llevaban a otros documentos, que permitía a los usuarios explorar, pero la experiencia resultaba limitada y lineal.

Con el paso del tiempo, los desarrolladores comenzaron a notar que la web necesitaba de dinamismo. Así que surgieron las paginas web generadas por el servidor que suponían un avance significativo ya que permitían crear contenido dinámico mediante la utilización de lenguajes como PHP, permitiendo a los desarrolladores crear páginas que se adaptasen a las necesidades del usuario. Pero la verdadera magia llegó al incorporar JavaScript, ya que esto permitía que al hacer clic en un botón o interactuar con un elemento, la pagina respondiese de inmediato.

La transformación continuo con la llegada de AJAX. Este conjunto de tecnologías proporcionaba una forma diferente para comunicarse con el servidor ya que las páginas podían enviar y recibir datos del servidor en segundo plano sin necesidad de recargar la página completa, permitiendo así a los

usuarios disfrutar de una experiencia mas fluida y dinámica ya que cuando el usuario interactúa la aplicación carga contenido adicional sin interrupciones.

A medida que exploramos las diferentes opciones, consideramos la relación entre el servidor y el cliente. Generar HTML y estilos en el servidor puede ser más fácil para aplicaciones pequeñas, pero en proyectos más grandes, esta estrategia puede llevar a la sobrecarga del servidor y complicar la organización del código. Por ello, en nuestro módulo de desarrollo, decidimos enfocarnos en las SPAs. Esta elección brinda a los estudiantes la oportunidad de aprender sobre la construcción de elementos del DOM, la comunicación asíncrona con el servidor y la gestión de eventos, permitiendo una formación más completa.

Hoy en día, la web se ha diversificado en varias categorías de aplicaciones:

- **Páginas web:** Contenido estático y dinámico accesible a través de navegadores. Son informativas y limitadas en interactividad.
- **Aplicaciones web:** Pueden ser SPAs o no. Tienen un enfoque más interactivo y permiten realizar operaciones complejas, manejando datos y estados de usuario.
- **Aplicaciones web responsivas:** Están diseñadas para adaptarse a diferentes dispositivos y tamaños de pantalla, mejorando la experiencia del usuario en móviles y computadoras.
- **PWA (Progresive Web App):** Combinan lo mejor de las páginas web y las aplicaciones móviles, permitiendo funcionalidades como instalación en el dispositivo y funcionamiento offline.
- **Apps Híbridas:** Utilizan frameworks como Ionic o Flutter para crear aplicaciones que funcionan en múltiples plataformas, combinando características de aplicaciones nativas y web.

5. Enlaces interesantes para investigación:

- Guía con tecnologías para convertirte en un desarrollador de frontend completo: https://roadmap.sh/frontend
- Guía de desarrollo de NodeJS: https://lopegonzalez.es/libros/desarrollo-web-en-entorno-cliente/unidad-1-seleccion-de-arquitecturas-y-herramientas-de-programacion/introduccion-a-node-js/
- Guía de inicialización con Git: https://lopegonzalez.es/libros/desarrollo-web-en-entorno-cliente/unidad-1-seleccion-de-arquitecturas-y-herramientas-de-programacion/introduccion-a-git/