DESARROLLO

FRONT END

-BÁSICO

Autor de contenido
Sandra Milena Guevara Morales

EMPEZAR

Tabla de Contenido

Presentación

El principal objetivo de este curso es proporcionar las habilidades necesarias a los estudiantes para crear frontends eficientes, seguros y teniendo en cuenta las mejores prácticas garantizando su escalabilidad, mantenibilidad, portabilidad y usabilidad.

Este curso abarca conceptos básicos como HTML, CSS, Javascript, pasando por tópicos más avanzados como el uso de la librería React para crear Single Page Applications. Adicionalmente, se aplicarán conceptos de autenticación y autorización de usuarios en una aplicación web para garantizar desde el frontend que los usuarios tienen acceso a las secciones permitidas estableciendo un control de acceso a la información.

Al finalizar, el curso cuenta con dos módulos que proporcionan los conceptos clave para realizar despliegues de aplicaciones web en Vercel y para el manejo de control de versiones de código. Este último resulta muy útil para trabajar no solo con las tecnologías del dominio de este curso, si no que podría aplicarse en el desarrollo de software en general con cualquier lenguaje de programación o cualquier tipo de framework o librería.

Objetivos del curso (competencias)

Objetivo general

Desarrollar aplicaciones frontend eficientes, seguras y teniendo en cuenta las mejores prácticas garantizando su escalabilidad, mantenibilidad, portabilidad y usabilidad.

Objetivo específico

- Aumentar la sensibilidad por la experiencia de los usuarios al navegar en una aplicación web.
- Establecer un primer acercamiento con el área de desarrollo de software aportando valor al usuario desde el primer día construyendo interfaces gráficas interactivas y funcionales.
- Implementar diferentes herramientas para la creación de una single page application.

Desarrollo del contenido

Unidad 1: Introducción al desarrollo web

1.1. Arquitectura cliente-servidor

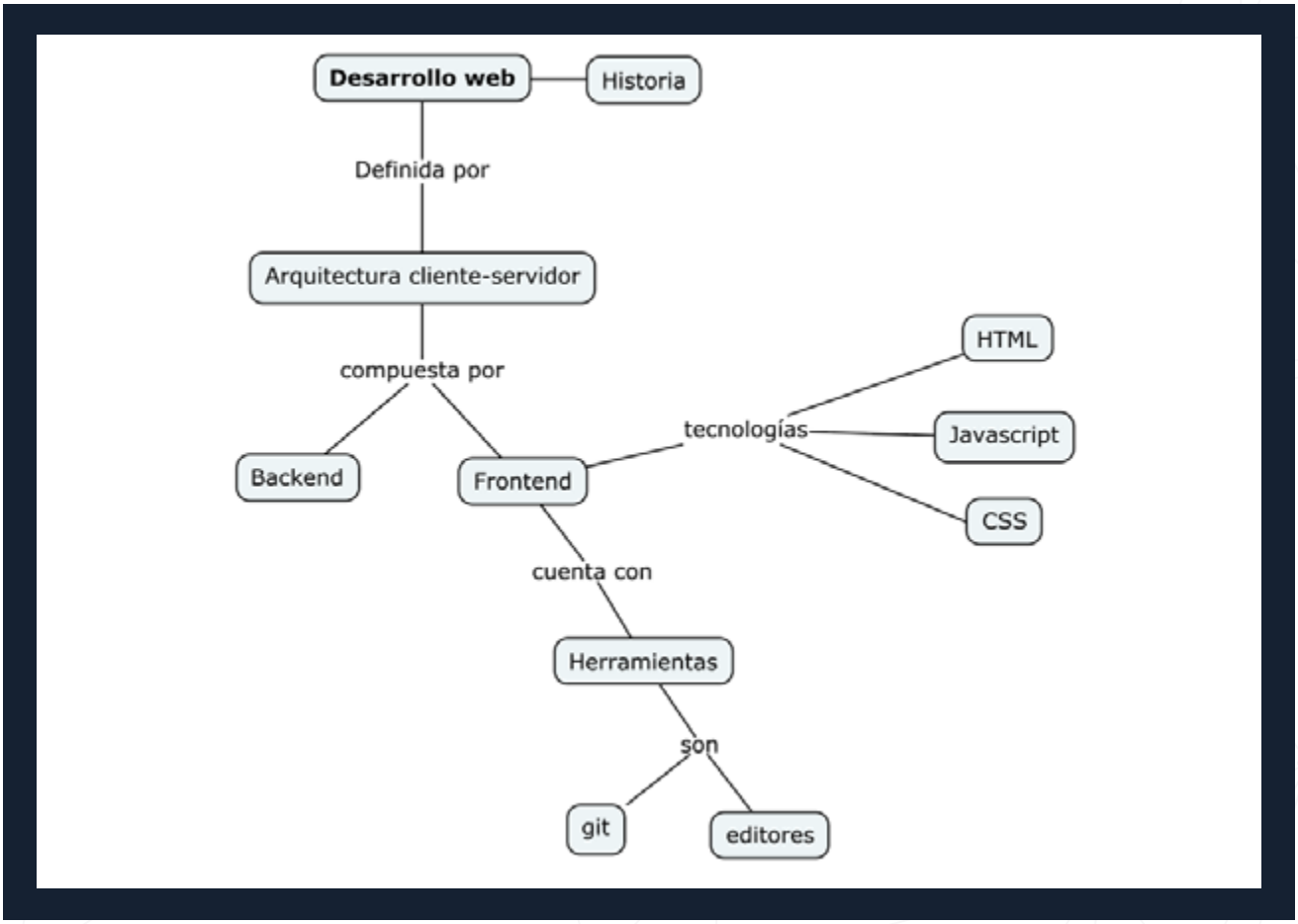
1.2. ¿Qué es HTML?

1.3. ¿Qué es CSS?

1.4. ¿Qué es Javascript?

1.5. Herramientas para el desarrollo frontend

Mapa de contenido de la unidad



1.1. Arquitectura cliente-servidor

Programación web

El desarrollo o programación web se refiere a la creación, construcción y mantenimiento de sitios web. Es la creación de una aplicación que funciona a través de Internet, es decir, sitios web.

Lenguaje de programación

Un lenguaje de programación es un lenguaje informático que los programadores utilizan para desarrollar programas de software, scripts u otros conjuntos de instrucciones para que los ejecuten las computadoras. Aunque muchos lenguajes comparten similitudes, cada uno tiene su propia sintaxis. Una vez que un programador aprende las reglas, la sintaxis y la estructura del lenguaje, escribe el código fuente en un editor de texto o IDE. Luego, el programador a menudo compila el código en un lenguaje de máquina que la computadora puede entender. Los lenguajes de secuencias de comandos, que no requieren un compilador, utilizan un intérprete para ejecutar la secuencia de comandos.

Historia

Los sistemas cliente-servidor empezaron a emerger en Estados Unidos cerca a los 80s cuando la informática pasó de grandes mainframes a procesamiento distribuido utilizando múltiples estaciones de trabajo o computadoras personales. Las empresas adoptaron rápidamente los sistemas cliente-servidor, que se convirtieron en la columna vertebral de su infraestructura de comunicaciones y automatización de oficinas.

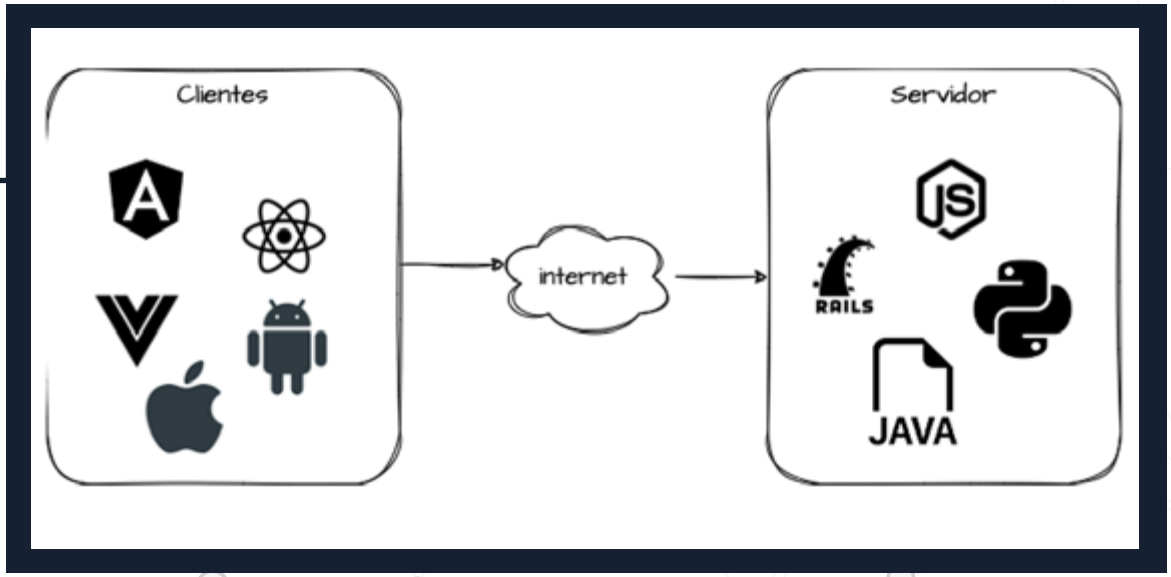
¿Qué es la arquitectura cliente servidor?

La arquitectura cliente-servidor define una aplicación distribuida que divide la carga de trabajo entre los proveedores de un recurso o servicio (servidores) y los solicitantes del servicio (clientes). Cuando el computador del cliente envía una solicitud de datos al servidor a través de internet, el servidor acepta el proceso solicitado y entrega los paquetes de datos solicitados al cliente.

Los clientes no comparten ninguno de sus recursos. Ejemplos de modelo cliente-servidor son correo electrónico, World Wide Web, etc.

Backend y Frontend

Cuando se escucha hablar de backend, generalmente se refiere al lado del servidor. Se llama “backend” porque el usuario no ve esta parte, ya que solo ve la interfaz de usuario de la aplicación. Siempre que sirva una respuesta al dispositivo solicitante, se puede llamar como servidor. Lo contrario de es frontend (lado del cliente), esto es lo que ve el usuario, por ejemplo: la interfaz de usuario de la aplicación móvil o la página web que se muestra en su navegador web. Esta interfaz gráfica es implementada por medio de tecnologías como: HTML, CSS y Javascript. La comunicación entre cliente-servidor se da por medio del protocolo HTTP (Hypertext Transfer Protocol) y estableciendo contratos en formato JSON (JavaScript Object Notation).



Arquitectura Cliente - Servidor

Bases de datos

Frameworks frontend:

- AngularJS
- React.js
- VueJS
- jQuery
- Bootstrap
- Material UI
- Tailwind CSS

Frameworks Backend

- Django
- Flask
- Ruby on Rails
- Spring
- Express
- Asp.Net Core

Protocolo HTTP

El Protocolo de transferencia de hipertexto (HTTP) es la base de la World Wide Web y se utiliza para cargar páginas web mediante enlaces de hipertexto. HTTP es un protocolo de capa de aplicación diseñado para transferir información entre dispositivos en red y se ejecuta sobre otras capas de la pila de protocolos de red. Un flujo típico a través de HTTP implica que una máquina cliente realice una solicitud a un servidor, que luego envía un mensaje de respuesta.

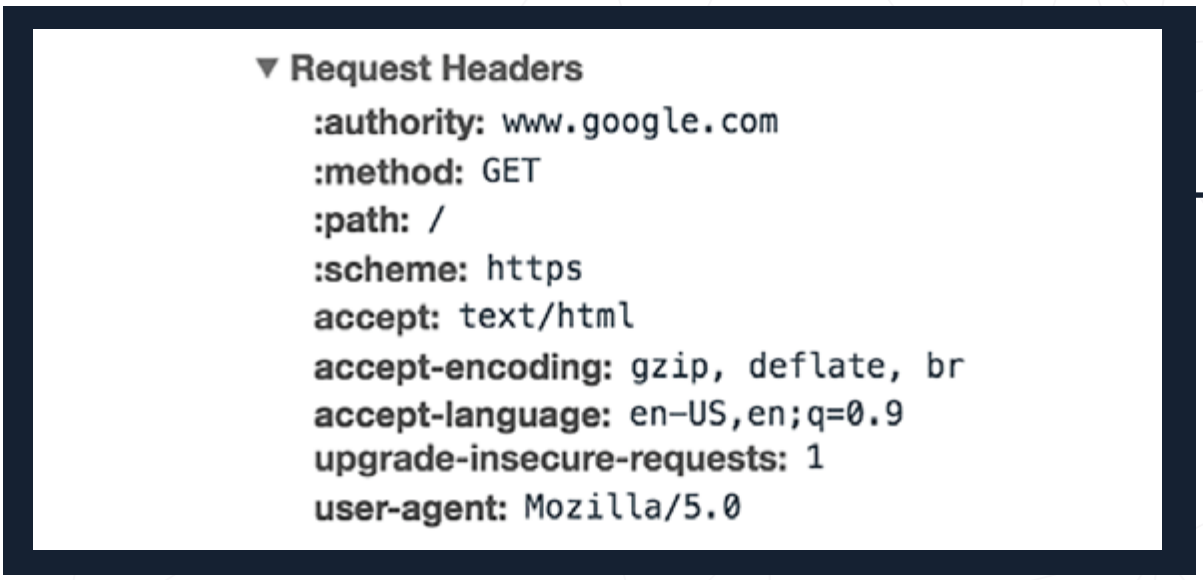
Una solicitud HTTP es la forma en que las plataformas de comunicaciones de Internet, como los navegadores web, solicitan la información que necesitan para cargar un sitio web.

Cada solicitud HTTP realizada a través de Internet lleva consigo una serie de datos codificados que transportan diferentes tipos de información. Una solicitud HTTP típica contiene:

- Versión HTTP
- URL
- Un método HTTP
- Headers HTTP
- Cuerpo HTTP (opcional).

- Método HTTP: Un método HTTP, a veces denominado verbo HTTP, indica la acción que la solicitud HTTP espera del servidor consultado. Por ejemplo, dos de los métodos HTTP más comunes son 'GET' y 'POST'; una solicitud 'GET' espera recibir información a cambio (generalmente en forma de un sitio web), mientras que una solicitud 'POST' generalmente indica que el cliente está enviando información al servidor web).

- Headers: Los encabezados HTTP contienen información de texto almacenada en pares clave-valor, y se incluyen en cada solicitud HTTP (y respuesta, más sobre eso más adelante). Estos encabezados comunican información central, como qué navegador está usando el cliente y qué datos se solicitan.



Ejemplo headers HTTP en Chome

- Body: El cuerpo de una solicitud es la parte que contiene el “cuerpo” de información que la solicitud está transfiriendo. El cuerpo de una solicitud HTTP contiene cualquier información que se envíe al servidor web, como un nombre de usuario y una contraseña, o cualquier otro dato ingresado en un formulario.

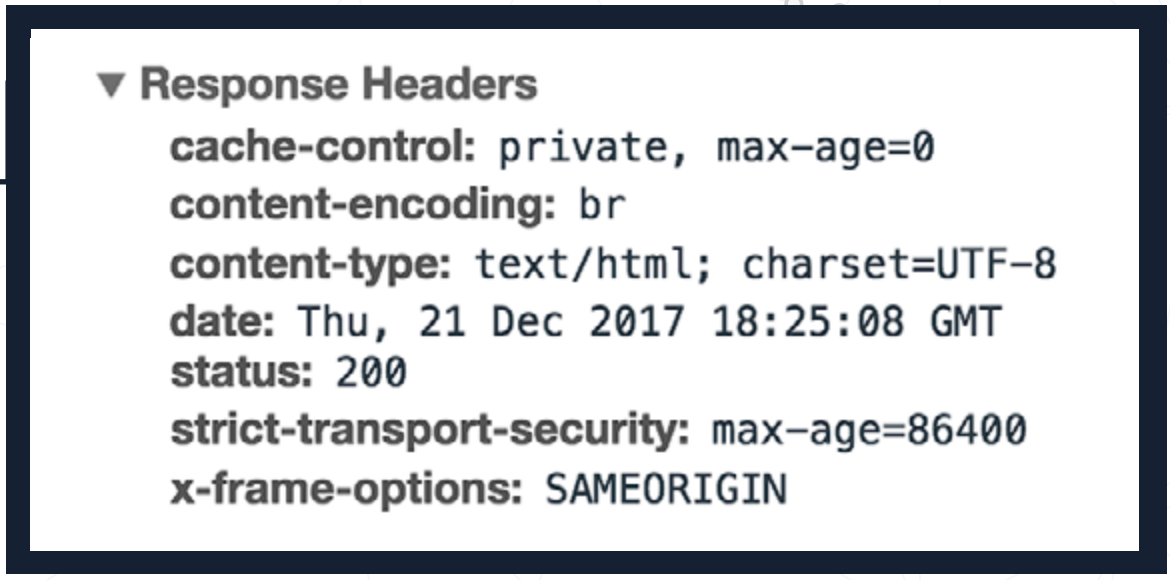
- Response: Una respuesta HTTP es lo que los clientes web (a menudo navegadores) reciben de un servidor de Internet en respuesta a una solicitud HTTP. Estas respuestas comunican información valiosa basada en lo que se solicitó en la solicitud HTTP.

- Status code: Los códigos de estado HTTP son códigos de 3 dígitos que se usan con mayor frecuencia para indicar si una solicitud HTTP se completó con éxito. Los códigos de estado se dividen en los siguientes 5 bloques:

- 1xx Información
- 2xx Exitoso
- 3xx Redirección
- 4xx Error en cliente
- 5xx Error en el servidor

Donde “xx” se refiere a diferentes números entre 00 y 99.

-Response Headers: Al igual que una solicitud HTTP, una respuesta HTTP viene con encabezados que transmiten información importante, como el idioma y el formato de los datos que se envían en el cuerpo de la respuesta.



Ejemplo de response headers en Chrome

- Response Body: Las respuestas HTTP exitosas a las solicitudes 'GET' generalmente tienen un cuerpo que contiene la información solicitada.

API

Una interfaz de programación de aplicaciones, o API, permite a las empresas abrir los datos y la funcionalidad de sus aplicaciones a desarrolladores externos y socios comerciales, o a departamentos dentro de sus empresas. Esto permite que los servicios y productos se comuniquen entre sí y aprovechen los datos y la funcionalidad de los demás a través de una interfaz documentada. Los programadores no necesitan saber cómo se implementa una API; simplemente usan la interfaz para comunicarse con otros productos y servicios. Son útiles para: Incrementar la colaboración entre unidades de negocio y flexibilidad.

Protocolos

- SOAP: es un protocolo API creado con XML, que permite a los usuarios enviar y recibir datos a través de SMTP y HTTP. Con las API SOAP, es más fácil compartir información entre aplicaciones o componentes de software que se ejecutan en diferentes entornos o están escritos en diferentes lenguajes.

- XML-RP: es un protocolo que se basa en un formato específico de XML para transferir datos, mientras que SOAP utiliza un formato XML propietario.

- JSON-RPC: es un protocolo similar a XML-RPC, ya que ambos son llamadas a procedimientos remotos (RPC), pero este usa JSON en lugar de formato XML para transferir datos.

- REST: (Representational State Transfer) es un conjunto de principios de arquitectura de API web, lo que significa que no existen estándares oficiales (a diferencia de los que tienen un protocolo). Para ser una API REST (también conocida como API RESTful), la interfaz debe cumplir con ciertas restricciones arquitectónicas.

Rest API

Una API, o interfaz de programación de aplicaciones, es un conjunto de reglas que definen cómo las aplicaciones o dispositivos pueden conectarse y comunicarse entre sí. Una API REST es una API que se ajusta a los principios de diseño de REST, o estilo arquitectónico de transferencia de estado de representación. Por esta razón, las API REST a veces se denominan API RESTful.

Definido por primera vez en 2000 por el científico informático Dr. Roy Fielding en su tesis doctoral, REST proporciona un nivel relativamente alto de flexibilidad y libertad para los desarrolladores. Esta flexibilidad es solo una de las razones por las que las API REST han surgido como un método común para conectar componentes y aplicaciones.

Algunas API, como SOAP o XML-RPC, imponen un marco estricto a los desarrolladores. Pero las API REST se pueden desarrollar utilizando prácticamente cualquier lenguaje de programación y admiten una variedad de formatos de datos. El único requisito es que se alineen con los siguientes cinco principios de diseño REST, también conocidos como restricciones arquitectónicas:

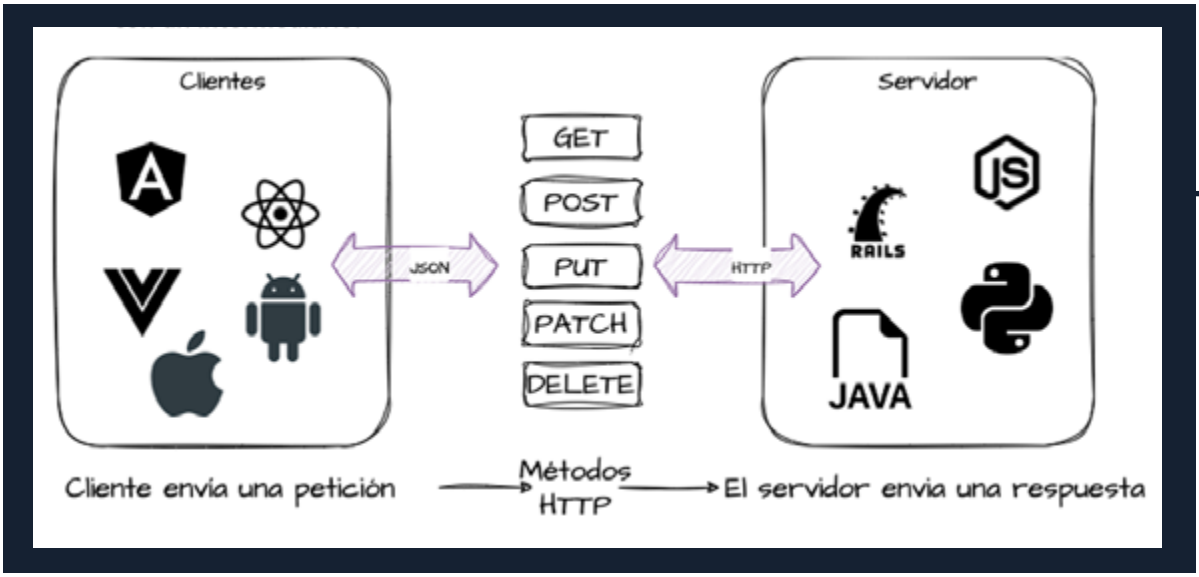
1. Interfaz uniforme: Todas las solicitudes de API para el mismo recurso deben tener el mismo aspecto, sin importar de dónde provenga la solicitud. La API REST debe garantizar que el mismo dato, como el nombre o la dirección de correo electrónico de un usuario, pertenezca a un solo identificador uniforme de recursos (URI). Los recursos no deben ser demasiado grandes, pero deben contener toda la información que el cliente pueda necesitar.

2. Desacoplamiento cliente-servidor. En el diseño de API REST, las aplicaciones de cliente y servidor deben ser completamente independientes entre sí. La única información que debe conocer la aplicación cliente es el URI del recurso solicitado; no puede interactuar con la aplicación del servidor de ninguna otra manera. De manera similar, una aplicación de servidor no debe modificar la aplicación de cliente más que pasarla a los datos solicitados a través de HTTP.

3. Statelessness: Las API REST no tienen estado, lo que significa que cada solicitud debe incluir toda la información necesaria para procesarla. En otras palabras, las API REST no requieren ninguna sesión del lado del servidor. Las aplicaciones de servidor no pueden almacenar ningún dato relacionado con una solicitud de cliente.

4. Capacidad de almacenamiento en caché. Cuando sea posible, los recursos deben almacenarse en caché en el lado del cliente o del servidor. Las respuestas del servidor también deben contener información sobre si se permite el almacenamiento en caché para el recurso entregado. El objetivo es mejorar el rendimiento del lado del cliente, al tiempo que aumenta la escalabilidad del lado del servidor.

5. Arquitectura del sistema en capas. En las API REST, las llamadas y las respuestas pasan por diferentes capas. Como regla general, no asuma que las aplicaciones cliente y servidor se conectan directamente entre sí. Puede haber varios intermediarios diferentes en el ciclo de comunicación. Las API REST deben diseñarse de modo que ni el cliente ni el servidor puedan saber si se comunica con la aplicación final o con un intermediario.



Flujo de peticiones entre cliente servidor con API REST

1.2. ¿Qué es HTML?

HTML (Lenguaje de etiquetas de hipertexto, en inglés HyperText Markup Language) es un lenguaje de marcado que se utiliza para el desarrollo de sitios web. Los navegadores son los encargados de interpretar el código HTML y mostrarlo en la forma que lo hayamos desarrollado.

Es tan importante y necesario porque es el lenguaje que interpretan los navegadores actuales como Firefox, Chrome, Safari, Edge, etc.

Estos navegadores se rigen con los estándares de la W3C, lo cual nos permite realizar un desarrollo y estar seguros podrá ser interpretado en cualquier navegador. HTML lo encontramos en cada una de las webs que visitamos asiduamente.

Utilizamos tags o etiquetas para dar estructura a nuestro contenidos dentro de un sitio web.

1.3. ¿Qué es CSS?

CSS - Cascading Style Sheets nace de la necesidad de darle formato a las etiquetas HTML. Hoy en día cada una de las etiquetas que tiene un sitio web tiene al menos alguna regla de código de CSS aplicado.

Su implementación es bastante visible e importante en cada uno de los sitios web, debido a la necesidad de posicionar elementos o tags, de realizar alineaciones y aplicar fuentes, entre otros.

1.4. ¿Qué es Javascript?

Antes de definir Javascript es necesario hablar de Document Object Model o DOM, que es la herramienta que ha influido en el desarrollo de las páginas web dinámicas y aplicaciones más complejas ya que nos permite acceder y manipular los elementos del documento html.

Ahora, Javascript es un lenguaje de programación que permite implementar, actualizar y controlar dinámicamente el DOM de una página web.

1.5. Herramientas para el desarrollo frontend

Git

Git es un sistema de control de versiones diseñado y desarrollado por Linus Torvalds. Con git es posible gestionar versiones de software y trabajar en equipo sin preocuparse por los conflictos que puedan existir cuando dos personas editan el mismo archivo.

Editores de código o IDE (Entorno integrado de desarrollo)

- Webstorm
- VSCode

Complementos Chrome

- Herramientas de desarrollador de google
- Para trabajo con react: React Developer Tools
- Para trabajo con Redux: Redux DevTools

Otros materiales para profundizar

Recursos de video

- render2web. (2021, 2 febrero). *Qué es Cliente y Servidor - Cliente Servidor Http Client-Server Video* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=4IAgB-6oAT0

- Victor Robles WEB. (2020, 8 marzo). *Aprende HTML en 15 Minutos* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=mNbnV3aN3KA

- Victor Robles WEB. (2020b, marzo 29). *Aprende CSS en 15 Minutos* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=3yM5uXp-T_0

- adrformacion. (2019, 10 octubre). *Curso de Control de Versiones con Git/GitHub* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=wsPthUULne0

- Programador X. (2021, 10 diciembre). *Ruta para ser programador BACKEND* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=fkVgNf3kMTM

- Programador X. (2021a, noviembre 19). *Ruta para ser programador FRONTEND* [Vídeo]. YouTube.
https://www.youtube.com/watch?v=lCGoeEF8T_I

Material complementario

- *Qué es HTML.* (2001, 1 enero). Desarrollo Web.
https://desarrolloweb.com/articulos/que-es-html.html

- *Qué es HTML*. **(s. f.).** CódigoFacilito. https://codigofacilito.com/articulos/que-es-html

- KeepCoding, R. (2022, 18 agosto). *Front End Back End y Full Stack, 3 tipos de developer codiciados*. KeepCoding Tech School. https://keepcoding.io/blog/frontend-backend-y-full-stack-3-tipos-developer/

- *Git - Acerca del Control de Versiones*. (s. f.). https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones

- Santos, D. (2022, 9 agosto). *Introducción al CSS: qué es, para qué sirve y otras 10 preguntas frecuentes*. https://blog.hubspot.es/website/que-es-css

Referencias bibliográficas de la unidad

- Sommerville, I. (2005). Ingeniería del software. Pearson Educación

- Tema 2: EL MODELO CLIENTE/SERVIDOR. (s. f.). E. U. *Informática en Segovia*.

- *Client-Server Web Apps with JavaScript and Java: Rich, Scalable, and RESTful*. **(2014).** Casimir Saternos.

- Comer, D., & Stevens, D. L. (1993). *Internetworking with TCP/IP.: Client-server Programming and Applications; Douglas E. Comer and David L. Stevens*. Prentice Hall.

- Luna, A. C. (2019). *Creación de páginas web*: HTML 5. ICB, SL (Interconsulting Bureau SL).

- Durango, A. (2015). *Diseño Web con CSS: 2ª Edición*. IT Campus Academy.

ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN

ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA

BOGOTÁ

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad