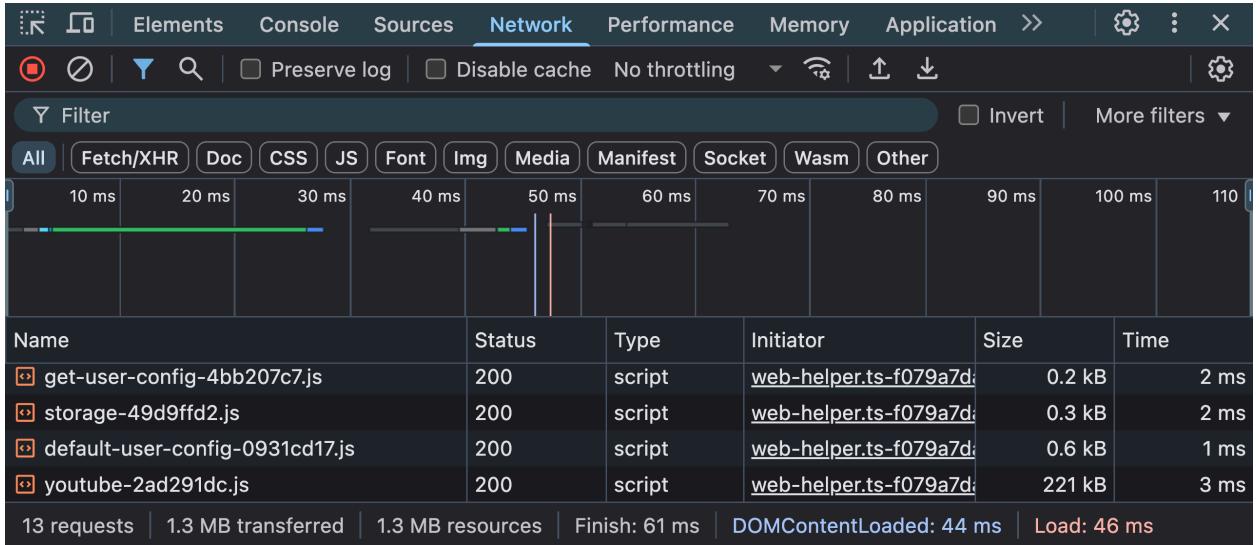
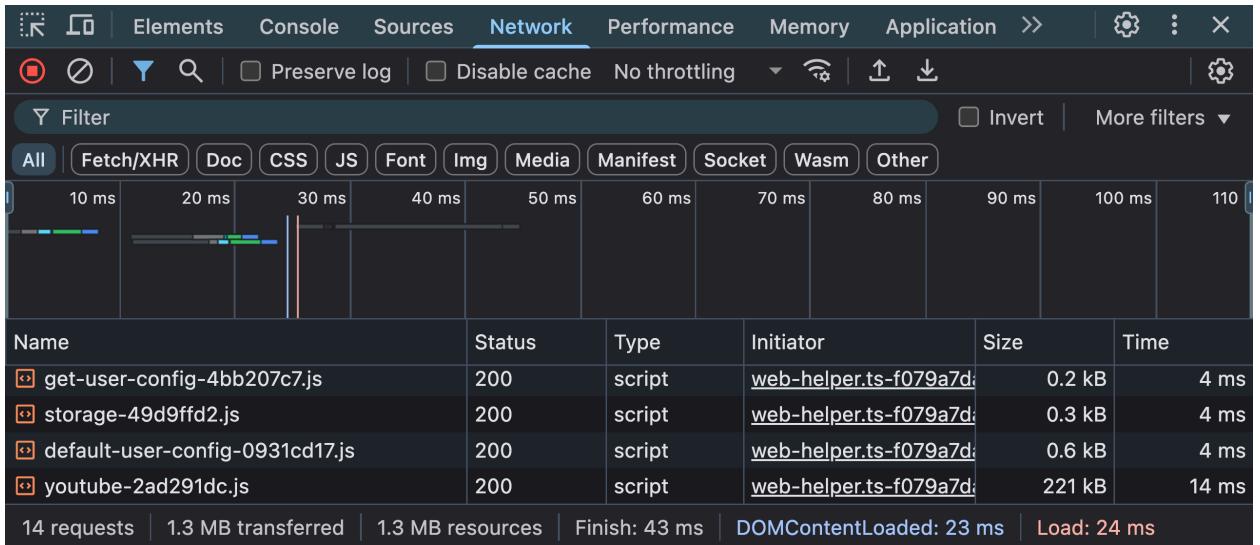


Ejercicio página01.html

- Sin refactorizar



- Factorizado



1. Importancia de la Separación de Responsabilidades

Aprendí que separar el HTML, JavaScript y CSS en archivos diferentes permite que el proyecto sea mucho más ordenado, fácil de entender y de mantener. Al modularizar, cada tecnología cumple su función específica:

- HTML estructura el contenido.
- JavaScript maneja la lógica y las interacciones.
- CSS controla la presentación visual.

Esto facilita el trabajo colaborativo y disminuye los errores al modificar el sistema en el futuro.

2. Mejora en el Rendimiento y Tiempos de Carga

Al dividir los archivos, la carga inicial de la página se optimiza. Los navegadores pueden cargar los recursos de manera más eficiente, y, gracias a prácticas como el uso de defer en el script, se evita que el JavaScript bloquee el renderizado de la página.

Esta mejora impacta directamente en la experiencia del usuario, haciendo que el sitio sea más rápido y fluido.

3. Buenas Prácticas de Programación

Comprendí la importancia de:

- Usar DOMContentLoaded para asegurarme de que el DOM esté listo antes de manipularlo.
- Manejar correctamente los eventos (addEventListener) en lugar de mezclar eventos directamente en atributos HTML (onclick).
- Utilizar funciones más seguras y modernas comotextContent en vez de childNodes[0].nodeValue.

Estos cambios permiten escribir un código más limpio, entendible y moderno.

4. Preparación para Escalabilidad

Refactorizar pensando en módulos me enseñó que un proyecto pequeño puede crecer fácilmente. Si en el futuro quiero agregar más botones, más tablas o funcionalidades como búsquedas o filtros, el sistema modularizado lo permitiría sin desordenar el código existente.

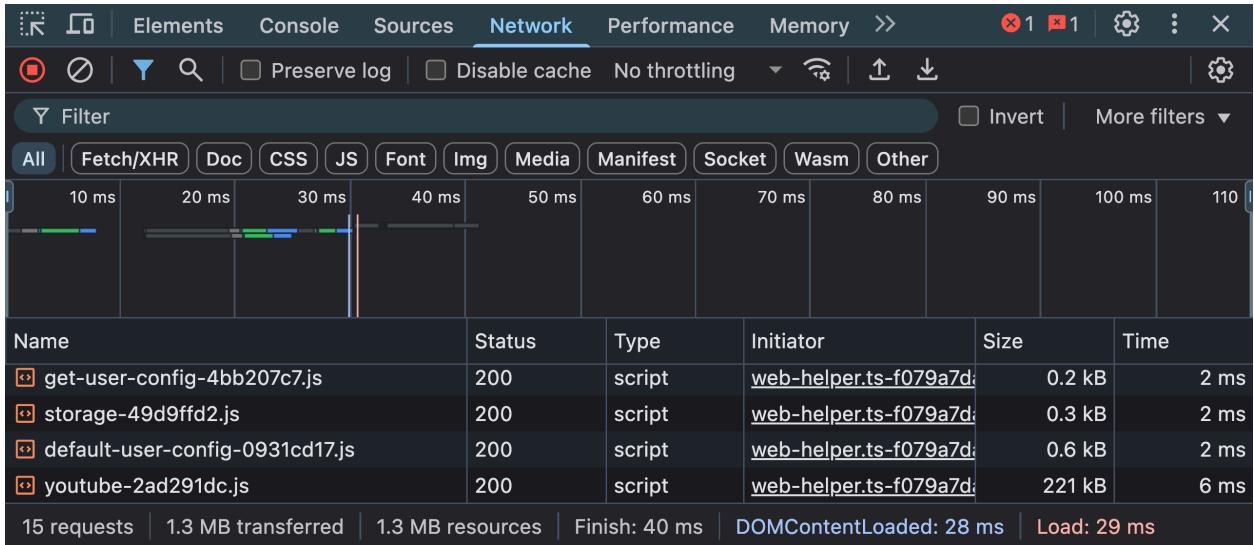
5. Responsabilidad como desarrollador

Finalmente, entendí que no basta con que un programa funcione: también debe estar bien estructurado, pensado para quienes darán mantenimiento o lo usarán más adelante.

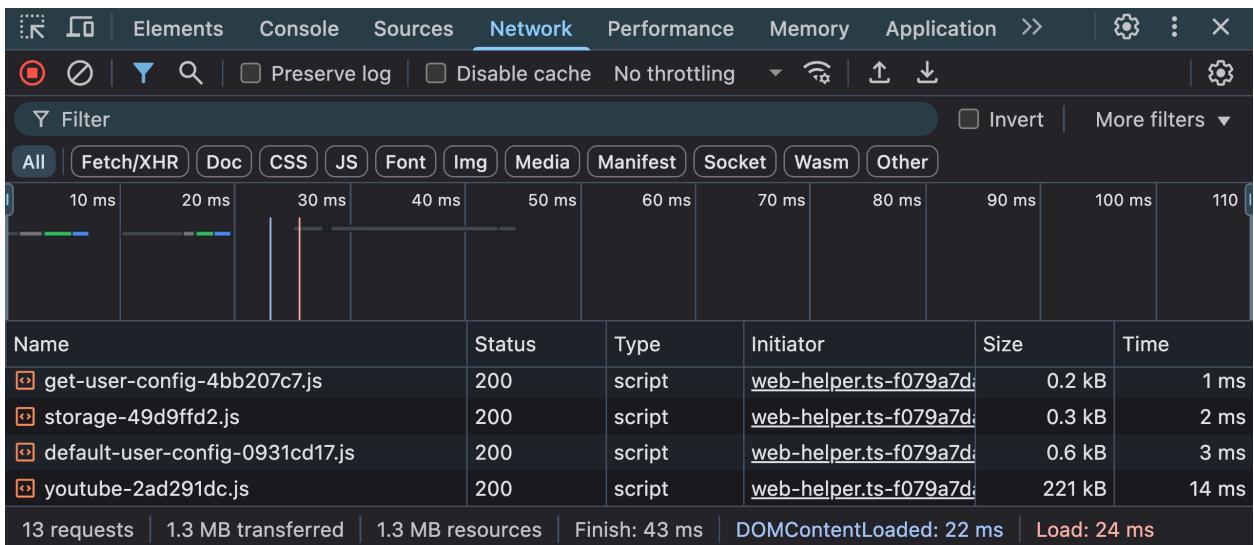
Esto es una parte clave de la responsabilidad profesional como desarrollador.

Ejercicio página02.html

- Sin refactorizar



- Factorizado



Durante el desarrollo de esta actividad, en la que refactoricé una página HTML que cargaba datos XML usando fetch y async/await, he obtenido varios aprendizajes importantes:

1. Uso Correcto de fetch y async/await

Aprendí a utilizar de forma correcta fetch para obtener archivos externos (en este caso, un XML), y cómo trabajar con async/await para manejar las operaciones asíncronas de una manera más limpia y entendible.

Además, entendí la importancia del manejo de errores usando try...catch para mejorar la robustez del código.

2. Importancia de los template literals en JavaScript

Durante la refactorización, detecté un error en la forma en que estaba construyendo dinámicamente el contenido HTML. Aprendí que para interpolar variables dentro de cadenas en JavaScript, es necesario utilizar template literals (delimitados por comillas invertidas `), lo cual hace que el código sea más claro y menos propenso a errores.

3. Separación de Archivos para una Mejor Organización

Refactorizar la página separando el HTML, el JavaScript y el CSS me permitió comprender la importancia de modularizar un proyecto:

- El HTML solo debe contener la estructura.
- El JavaScript debe manejar la lógica y la interacción.
- El CSS debe encargarse de la presentación visual.

Esta división facilita la lectura del código, su mantenimiento y su escalabilidad.

4. Buenas Prácticas de Carga de Scripts

Aprendí a usar el atributo defer en el <script> del HTML para asegurar que el navegador no bloquee la carga del contenido estructural de la página mientras

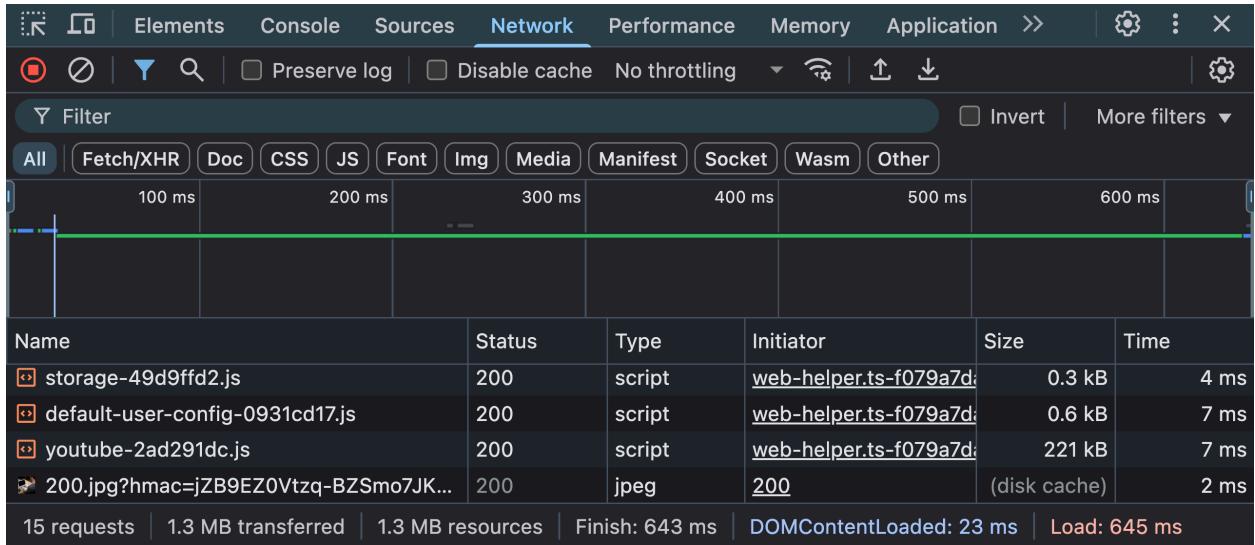
descarga y ejecuta el JavaScript. Esto contribuye a mejorar los tiempos de carga y la experiencia del usuario.

5. Preparación para Crecimiento del Proyecto

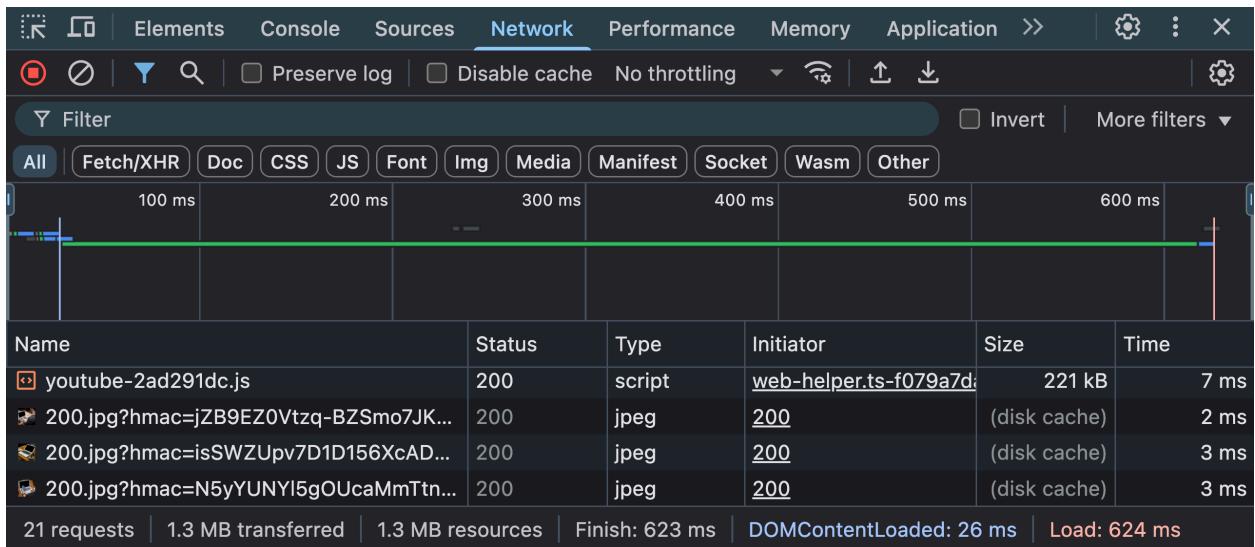
Comprendí que modularizar no solo organiza mejor el código, sino que también prepara el proyecto para su crecimiento futuro. Si se agregan más botones, tablas o nuevas funcionalidades, será mucho más sencillo integrarlas sin generar desorden.

Ejercicio pagina05.html

- Sin factorizar



- Factorizada



Durante esta actividad, donde trabajé en un generador de imágenes aleatorias utilizando JavaScript y HTML dinámico, logré varios aprendizajes importantes:

1. Uso de Template Literals para Construir Código HTML Dinámico

Aprendí que los template literals (comillas invertidas `) son la forma más eficiente y moderna para construir fragmentos de HTML dinámico, especialmente cuando se incluyen variables (\${ }) dentro del contenido.

Comparé la concatenación tradicional con + (en la función imagen) contra el uso de template literals (en la función Picture), y comprobé que el segundo método es más limpio, claro y menos propenso a errores.

2. Importancia de Separar Estructura, Estilos y Lógica

Refactorizar esta página me enseñó que, para un mejor mantenimiento y escalabilidad:

- El HTML debe centrarse únicamente en la estructura (contenedores y etiquetas básicas).
- El CSS debe definirse en un archivo separado para controlar la presentación y apariencia visual.
- El JavaScript debe residir en un archivo propio, manejando la generación de contenido y la interacción.

Esta separación modular permite trabajar de manera más organizada, encontrar errores más rápido y mejorar la velocidad de carga de los recursos.

3. Dinámica de Inserción de Elementos en el DOM

Aprendí cómo utilizar document.getElementById().innerHTML para insertar contenido dinámico en el DOM de manera sencilla.

También entendí la importancia de preparar correctamente el HTML generado (en formato válido) antes de insertarlo, para evitar errores de visualización.

4. Buenas Prácticas de Depuración

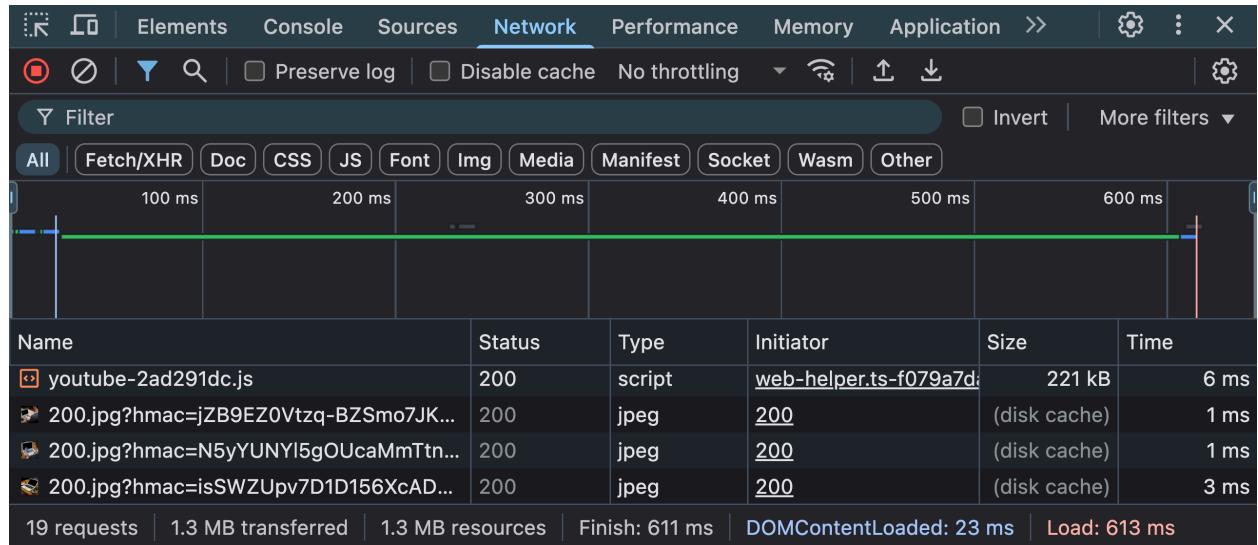
El uso de `console.log()` me permitió validar el estado de los elementos antes de manipularlos, una práctica fundamental para depurar código JavaScript de manera efectiva y evitar errores de ejecución.

5. Aplicación de Estilos Básicos en CSS

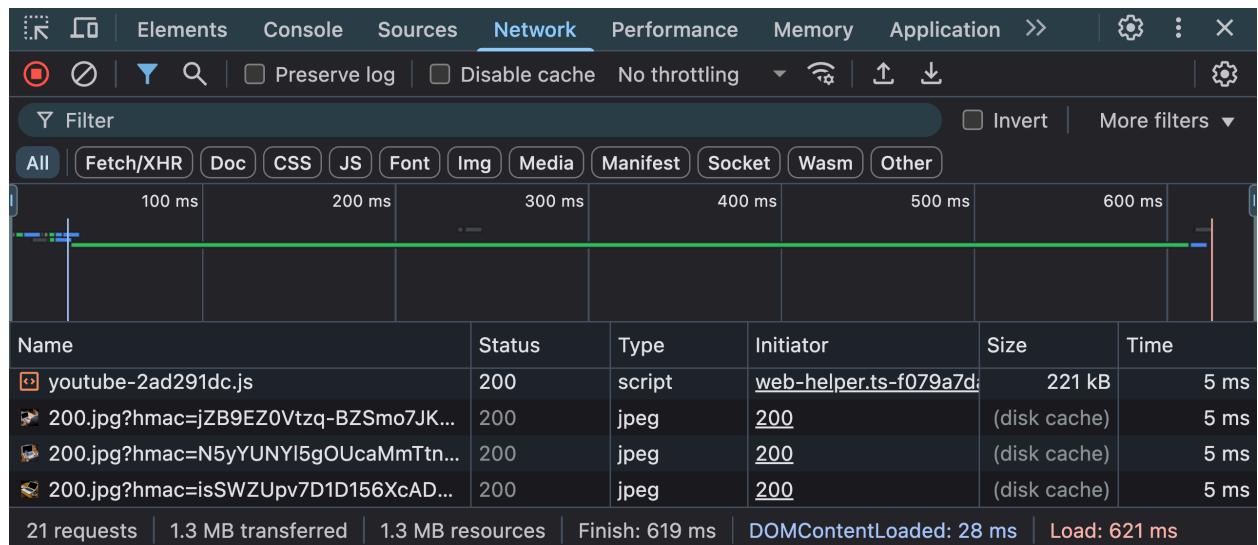
Entendí cómo aplicar estilos básicos en CSS para mejorar visualmente los componentes dinámicos, como agregar sombras a las imágenes y centrar las figuras, lo cual contribuye a una mejor experiencia de usuario.

Ejercicio pagina06.html

- Sin refactorizar



- Factorizada



Durante esta actividad, en la cual desarrollé y refactoricé una galería interactiva de imágenes, logré varios aprendizajes importantes:

1. Modularización Correcta de HTML, CSS y JavaScript

Aprendí que separar el HTML, CSS y JavaScript en archivos diferentes facilita mucho la organización del proyecto.

Cada archivo cumple una función específica:

- El HTML estructura el contenido.
- El CSS define el estilo visual y las transiciones.
- El JavaScript maneja la lógica dinámica y la interacción del usuario.

Esta modularización mejora el mantenimiento del proyecto y permite que el navegador cargue los recursos de manera más eficiente.

2. Generación Dinámica de Contenido

Comprendí cómo crear elementos HTML dinámicamente a través de funciones JavaScript usando template literals, lo cual permite insertar de forma más limpia y segura estructuras HTML que dependen de variables.

Esto es fundamental para desarrollar aplicaciones dinámicas y escalables en la web.

3. Manipulación del DOM y Eventos de Usuario

Aprendí a seleccionar múltiples elementos del DOM (`querySelectorAll`) y a asignar eventos (`addEventListener`) a cada uno de ellos.

En este caso, al hacer clic en una imagen, puedo cambiar su forma agregando o quitando una clase CSS (`toggle("redonda")`), lo cual mejora la interactividad del sitio.

4. Mejores Prácticas en la Estructura del Código

Comprendí la importancia de esperar al evento DOMContentLoaded antes de manipular el DOM para asegurar que todos los elementos estén correctamente cargados.

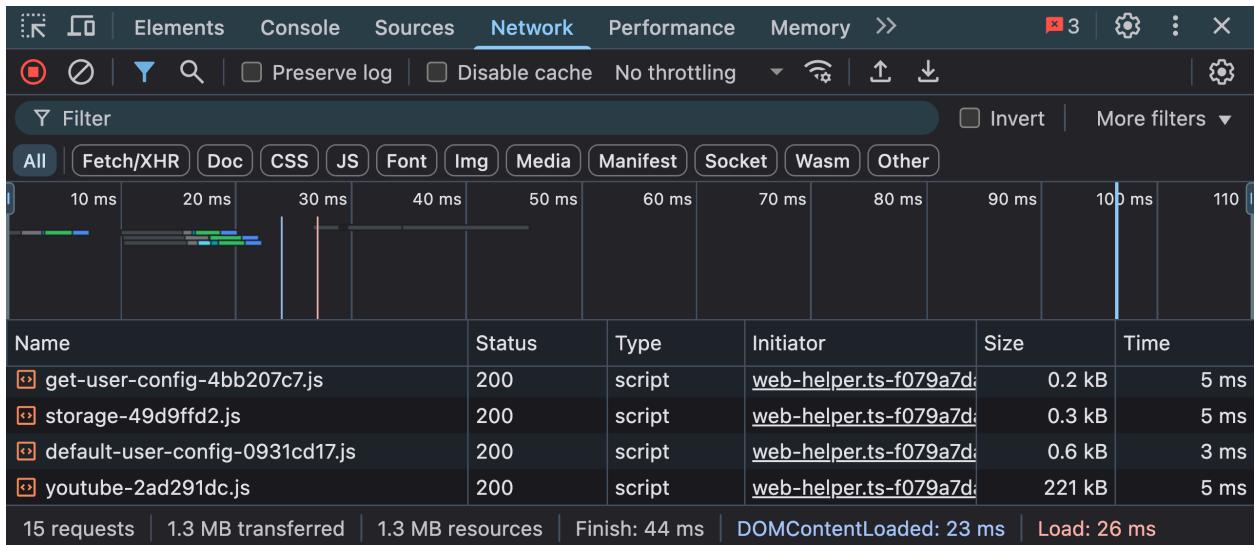
También aprendí a evitar prácticas como escribir `<script>` dentro de un `<div>` y a preferir cargar los scripts de forma externa usando `defer`.

5. Uso de Transiciones y Estilos Dinámicos en CSS

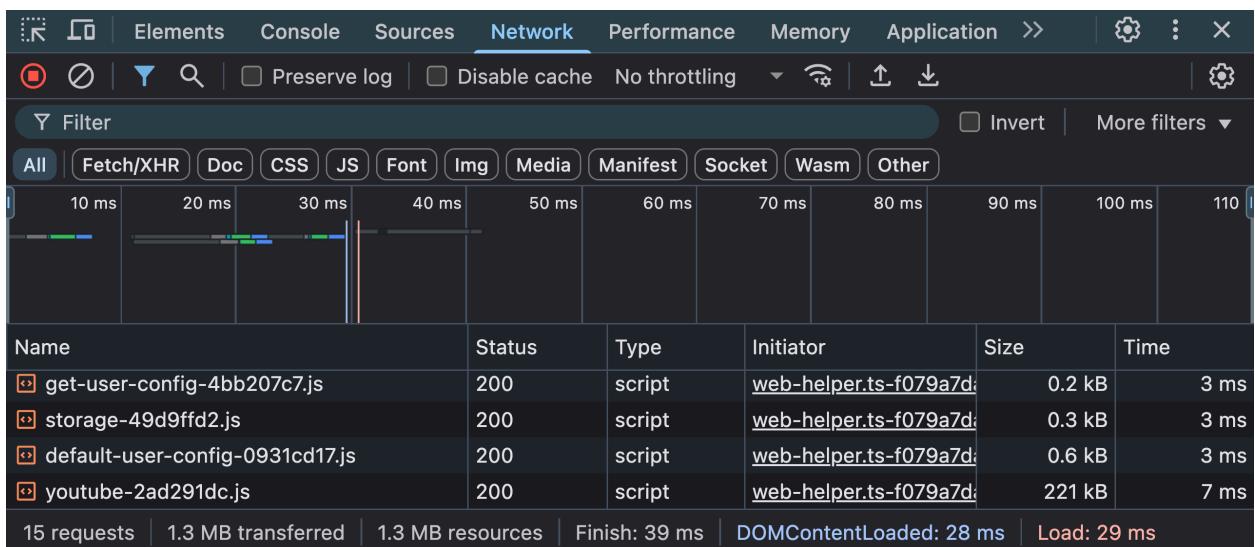
Aprendí a aplicar efectos visuales dinámicos mediante transiciones en CSS, logrando que el cambio de forma de las imágenes sea suave al hacer clic. Esto mejora notablemente la experiencia del usuario y da una apariencia más profesional a la aplicación.

Ejercicio pagina07.html

- Sin refactorizar



- Factorizado



Durante el desarrollo y refactorización de este formulario de inicio de sesión, obtuve varios aprendizajes importantes que me ayudaron a mejorar tanto la estructura como la funcionalidad de aplicaciones web básicas:

1. Modularización Correcta de Proyectos Web

Aprendí a separar adecuadamente un proyecto en archivos independientes:

- El HTML define únicamente la estructura y los campos visibles para el usuario.
- El CSS controla el diseño, los estilos visuales y la experiencia de interacción.
- El JavaScript maneja toda la lógica y los eventos dinámicos.

Esta modularización no solo organiza mejor el proyecto, sino que también facilita el mantenimiento y escalabilidad futura.

2. Manejo de Formularios con JavaScript

Reflexioné sobre la diferencia entre enviar formularios de forma tradicional (usando submit) y controlarlos manualmente con JavaScript:

- Capturando el evento del botón.
- Validando campos antes del envío.
- Modificando dinámicamente los atributos de los campos (name, action) para adaptarlos a distintas necesidades.

Este control total me permite construir formularios más seguros y adaptados a distintos flujos de trabajo.

3. Implementación de Validaciones Básicas

Implementé validaciones básicas para asegurar que el usuario no deje campos vacíos antes de enviar el formulario.

Entendí que las validaciones en el cliente son esenciales para mejorar la experiencia del usuario, prevenir errores y reducir cargas innecesarias en el servidor.

4. Buenas Prácticas en la Estructura del Código

Aprendí a utilizar DOMContentLoaded para asegurar que todos los elementos del DOM estén completamente cargados antes de ser manipulados.

También reforcé la importancia de usar addEventListener en lugar de eventos embebidos en el HTML, logrando un código más limpio y profesional.

5. Mejora de la Experiencia de Usuario mediante Diseño Visual

Gracias al uso de CSS bien estructurado:

- Logré un formulario centrado, amigable y visualmente atractivo.
- Apliqué animaciones suaves (hover, focus) para mejorar la interacción.
- Mejoré la accesibilidad utilizando adecuadamente label e input.

Estos detalles contribuyen significativamente a una mejor experiencia para el usuario final.