

# The Magic Library

HITO2\_2T\_DWEC\_ELOY



Por Eloy Pérez Gómez

# Índice

<b>P1 Identificar el propósito y los tipos de DNS, explicando cómo se organizan y gestionan los nombres de dominio.</b>	<b>3</b>
<b>P2 Explicar el objetivo y las relaciones entre diferentes protocolos de comunicación, hardware de servidor, sistemas operativos y software de servidores web en relación al diseño, publicación y acceso al sitio web.</b>	<b>5</b>
<b>P3 Analizar las capacidades y relaciones entre tecnologías web front-end y back-end y explicar cómo se relacionan respecto a las capas de presentación y aplicación.</b>	<b>7</b>
<b>P4 Discutir sobre las diferencias entre herramientas de creación web online y sitios personalizados en relación a flexibilidad de diseño, comportamiento, funcionalidad, User Experience (UX) y User Interface (UI).</b>	<b>9</b>
<b>P5 Documento de Diseño para The Magic Library - Sitio Web Multipágina</b>	<b>11</b>
<b>P6 - Estructura de Navegación</b>	<b>13</b>
<b>P7 - Testing Web</b>	<b>14</b>
<b>M1 Analizar el impacto de las tecnologías habituales de desarrollo web y frameworks en relación al diseño web, funcionalidad y gestión.</b>	<b>15</b>
<b>M2 Revisar la influencia de los motores de búsqueda en el comportamiento de los sitios web en base a su indexación y la optimización en dichos motores de búsqueda.</b>	<b>17</b>
<b>M3 - Analizar el conjunto de herramientas y técnicas disponibles para el diseño y desarrollo de un sitio web personalizado.</b>	<b>19</b>
<b>M4 Justifica las decisiones de implementación del sitio web respecto al design document.</b>	<b>23</b>
<b>M5 Analiza el proceso QA y review cómo fue implementado durante las etapas de diseño y desarrollo del sitio web.</b>	<b>25</b>
<b>D1 Justificar las tecnologías, servicios, herramientas y software elegidos para la realización del sitio web.</b>	<b>27</b>
<b>D2 Evalúa el diseño y el proceso de desarrollo del sitio web respecto al design document incluyendo algunos desafíos técnicos a los que te has enfrentado</b>	<b>28</b>
<b>D3 Evalúa los resultados del Test Plan y el éxito completo del sitio web con recomendaciones para su mejora</b>	<b>29</b>



# **P1 Identificar el propósito y los tipos de DNS, explicando cómo se organizan y gestionan los nombres de dominio.**

## **Propósito de DNS (Sistema de Nombres de Dominio):**

- Propósito Principal: Facilitar la traducción de nombres de dominio legibles por humanos a direcciones IP numéricas utilizadas por las computadoras en Internet.
- Facilitador de la Navegación: Permite a los usuarios acceder a sitios web mediante nombres de dominio en lugar de recordar direcciones IP complejas.

## **Tipos de DNS:**

### DNS Autoritativo:

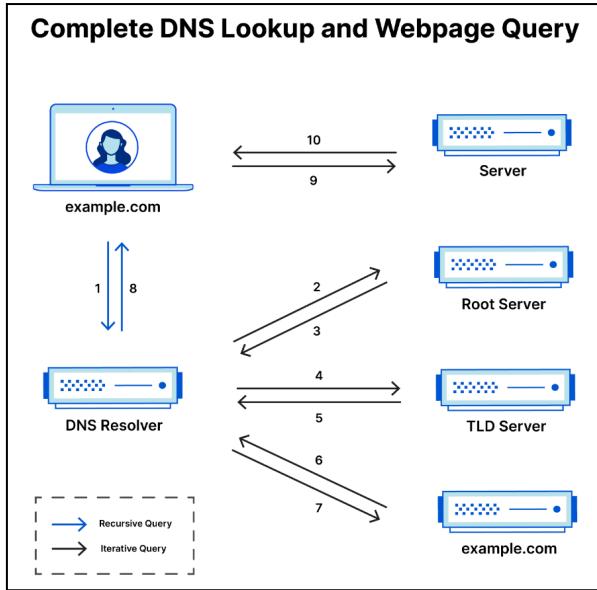
- Propósito: Almacena información oficial y actualizada sobre un dominio.
- Gestión: Mantenido por el administrador del dominio, contiene registros que definen la información sobre el dominio.

### DNS de Reenvío (Forwarding DNS):

- Propósito: Sirve como intermediario que reenvía consultas DNS a otros servidores DNS.
- Gestión: Utilizado para mejorar la eficiencia y velocidad de las consultas al reenviarlas a servidores DNS especializados.

### DNS de Caché:

- Propósito: Almacena temporalmente consultas y respuestas DNS para acelerar futuras solicitudes.
- Gestión: Mantiene en memoria las respuestas de consultas anteriores, reduciendo la necesidad de buscar la información cada vez.

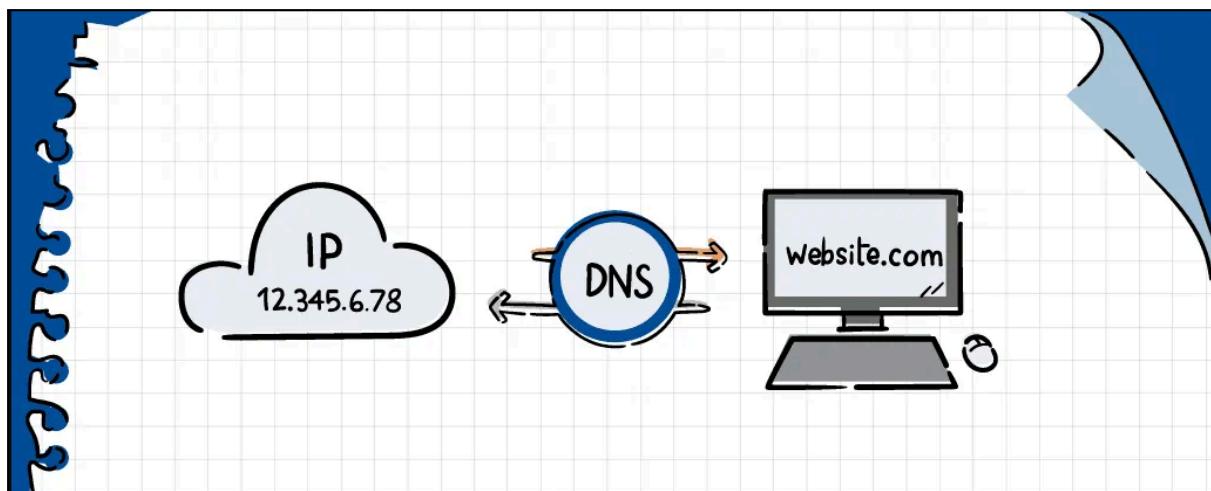


DNS nociones básicas

## Organización y Gestión de Nombres de Dominio:

- *Jerarquía de Dominio*: Los nombres de dominio se organizan en una estructura jerárquica, dividiéndose en niveles como subdominios y dominios de nivel superior (TLD).
- *Gestión Centralizada*: La gestión de nombres de dominio es descentralizada pero sigue ciertas normas establecidas por entidades como ICANN.
- *Servidores de Nombres (Name Servers)*: Almacenan y gestionan información de dominio, participando en la resolución de consultas DNS.

La organización y gestión eficientes del DNS son esenciales para garantizar la resolución rápida y precisa de nombres de dominio, facilitando así la navegación en la web.



¿Qué es y cómo funciona?

## **P2 Explicar el objetivo y las relaciones entre diferentes protocolos de comunicación, hardware de servidor, sistemas operativos y software de servidores web en relación al diseño, publicación y acceso al sitio web.**

### **Objetivo:**

- **Propósito Fundamental:** Facilitar el diseño, publicación y acceso a sitios web.

### **Relaciones entre Componentes:**

#### Protocolos de Comunicación:

- **Objetivo:** Establecer reglas y normas para la transmisión de datos entre dispositivos.
- **Relación:** Los protocolos como HTTP/HTTPS rigen la comunicación entre el navegador del usuario y el servidor web, garantizando la entrega correcta de contenido.

#### Hardware de Servidor:

- **Objetivo:** Proporcionar recursos computacionales para almacenar y procesar solicitudes.
- **Relación:** El hardware, como servidores dedicados o en la nube, influye en la capacidad de respuesta y rendimiento del sitio web.

#### Sistemas Operativos:

- **Objetivo:** Gestionar recursos de hardware y proporcionar una interfaz para la ejecución de software.
- **Relación:** El sistema operativo del servidor, como Linux o Windows Server, afecta la compatibilidad y eficiencia del software del servidor web.

#### Software de Servidores Web:

- **Objetivo:** Gestionar solicitudes HTTP, servir contenido web y ejecutar aplicaciones.
- **Relación:** Herramientas como Apache, Nginx o Microsoft IIS se integran con el sistema operativo y manejan la lógica de procesamiento de solicitudes web.

#### **Interconexión y Coordinación:**

- La comunicación fluida entre protocolos, hardware, sistemas operativos y software de servidores web es esencial para un diseño web eficiente, una publicación sin problemas y un acceso rápido a los sitios web.

## Ejemplo Práctico:

- Un navegador (utilizando el protocolo HTTP/HTTPS) solicita una página web. El hardware del servidor procesa la solicitud con la ayuda del sistema operativo, que a su vez coordina con el software del servidor web para entregar la página solicitada de manera eficiente al navegador del usuario.

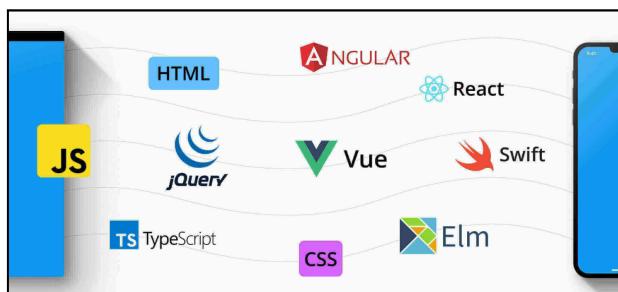


# P3 Analizar las capacidades y relaciones entre tecnologías web front-end y back-end y explicar cómo se relacionan respecto a las capas de presentación y aplicación.

## Capacidades y Relaciones:

### Tecnologías Front-end:

- **Capacidades:** Se centran en la interfaz de usuario y la presentación visual del sitio web.
- **Relaciones:** Están compuestas por HTML para la estructura, CSS para el estilo y JavaScript para la interactividad del usuario.



Tecnologías Frontend

### Tecnologías Back-end:

- **Capacidades:** Gestionan la lógica de la aplicación, la manipulación de datos y las operaciones del servidor.
- **Relaciones:** Incluyen lenguajes como Python, PHP, Ruby, y frameworks como Django, Laravel o Express para manejar la lógica del servidor y acceder a bases de datos.



Tecnologías Backend

## **Relación entre Front-end y Back-end:**

- Interacción de Capas: El front-end y el back-end trabajan juntos para proporcionar una experiencia completa al usuario.
- Comunicación mediante API: Las tecnologías back-end ofrecen puntos de acceso (API) que permiten la comunicación y transferencia de datos entre las capas.
- Separación de Responsabilidades:
  - El front-end se encarga de la presentación y la interfaz de usuario.
  - El back-end maneja la lógica de negocio, procesamiento de datos y almacenamiento.

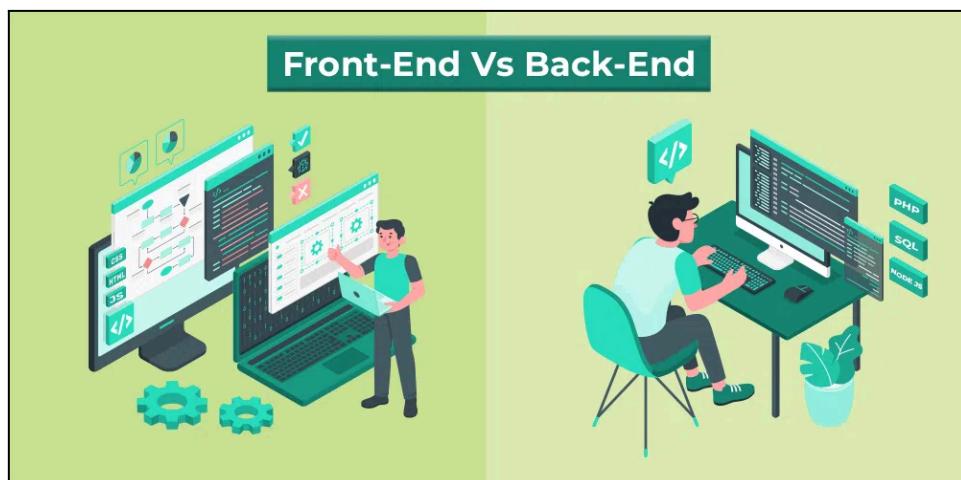
## **Ejemplo Práctico:**

- Un usuario interactúa con un formulario (front-end) para enviar datos. JavaScript procesa la interacción del usuario y hace una solicitud al back-end a través de una API. El back-end, utilizando lógica de servidor, procesa los datos, realiza operaciones en la base de datos y envía una respuesta al front-end, que actualiza la interfaz de usuario según sea necesario.

## **Beneficios de esta Relación:**

- Escalabilidad: Permite actualizar o cambiar el front-end sin afectar la lógica del back-end y viceversa.
- Eficiencia: Divide las responsabilidades, facilitando el desarrollo y la mantenibilidad.

Esta interrelación equilibrada entre front-end y back-end es esencial para crear sitios web robustos y dinámicos que ofrezcan una experiencia de usuario fluida y funcional.



FrontEnd VS BackEnd

## **P4 Discutir sobre las diferencias entre herramientas de creación web online y sitios personalizados en relación a flexibilidad de diseño, comportamiento, funcionalidad, User Experience (UX) y User Interface (UI).**

### **Herramientas de Creación Web Online:**

- Flexibilidad de Diseño:
  - **Ventajas:** Suelen ofrecer plantillas y diseños predefinidos que facilitan la creación rápida de sitios.
  - **Limitaciones:** Pueden tener restricciones en la personalización avanzada y la originalidad del diseño.
- Comportamiento y Funcionalidad:
  - **Ventajas:** Incorporan funciones preprogramadas y widgets para facilitar la inclusión de características específicas.
  - **Limitaciones:** Pueden carecer de flexibilidad para adaptarse a necesidades complejas o personalizadas.
- User Experience (UX) y User Interface (UI):
  - **Ventajas:** Suelen ofrecer interfaces intuitivas y amigables para usuarios sin experiencia técnica.
  - **Limitaciones:** La personalización puede ser limitada, lo que puede afectar la experiencia del usuario.

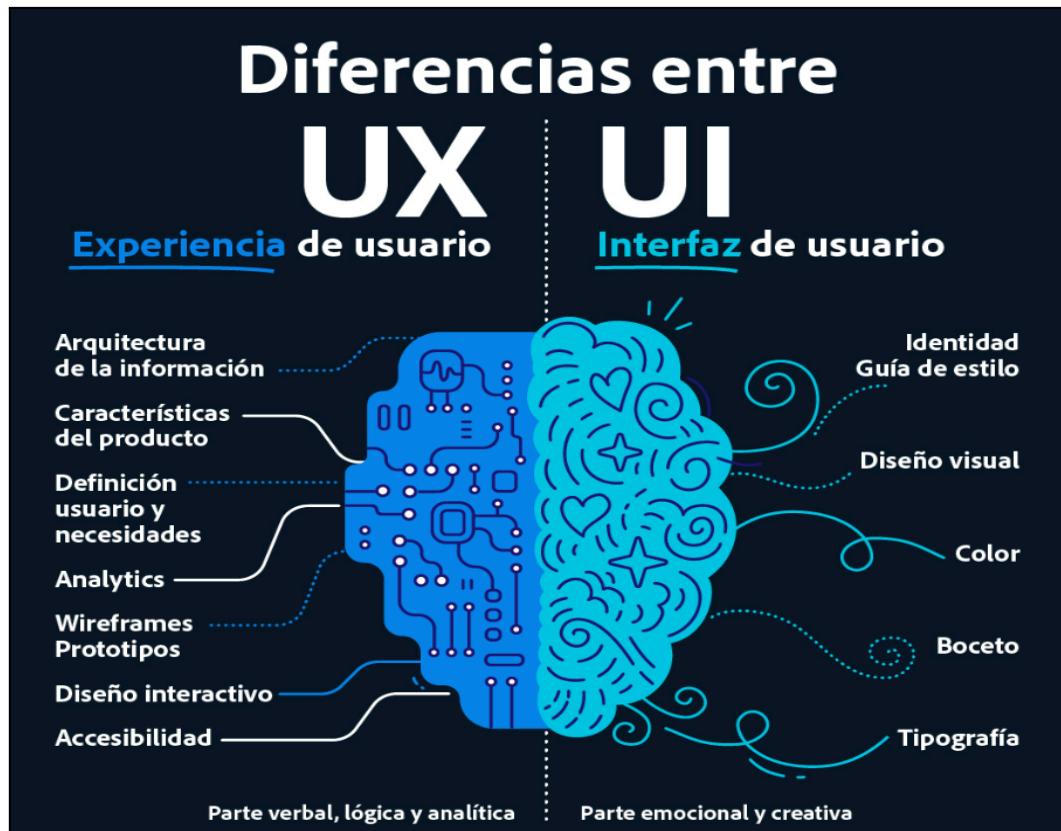
### **Sitios Personalizados:**

- Flexibilidad de Diseño:
  - **Ventajas:** Ofrecen libertad total para diseñar interfaces únicas y personalizadas.
  - **Limitaciones:** Requieren habilidades de diseño más avanzadas y tiempo para la creación desde cero.
- Comportamiento y Funcionalidad:
  - **Ventajas:** Permiten implementar funciones específicas y personalizadas según los requisitos del proyecto.
  - **Limitaciones:** Requieren un desarrollo más extenso para implementar funcionalidades específicas.
- User Experience (UX) y User Interface (UI):
  - **Ventajas:** Posibilidad de diseñar experiencias de usuario altamente adaptativas y específicas.
  - **Limitaciones:** La calidad de UX/UI depende en gran medida de las habilidades del diseñador y desarrollador.

## Consideraciones Generales:

- Proyectos Simples vs. Complejos:
  - Las herramientas online son eficaces para proyectos simples con requisitos estándar.
  - Sitios personalizados son preferibles para proyectos complejos con necesidades específicas.
- Tiempo y Costo:
  - Las herramientas online son rápidas y económicas para proyectos pequeños.
  - Sitios personalizados pueden requerir más tiempo y recursos, pero ofrecen mayor personalización.
- Escalabilidad:
  - Sitios personalizados son más escalables y pueden adaptarse al crecimiento futuro.
  - Las herramientas online pueden tener limitaciones a medida que los proyectos se vuelven más grandes y complejos.

La elección entre herramientas de creación web online y sitios personalizados depende de la naturaleza del proyecto, los requisitos específicos y las habilidades disponibles. Ambas opciones tienen su lugar en el desarrollo web, cada una con sus propias ventajas y limitaciones.



Diferencias UX UI

## P5 Documento de Diseño para The Magic Library - Sitio Web Multipágina

El **logo** que he decidido para el sitio web (y para mis proyectos en general), es el siguiente:



He diseñado un documento de diseño con **Figma**, creando una estructura con unas imágenes que tengo elegidas ya. Quiero que la estructura de mi web se asemeje a la siguiente estructura.

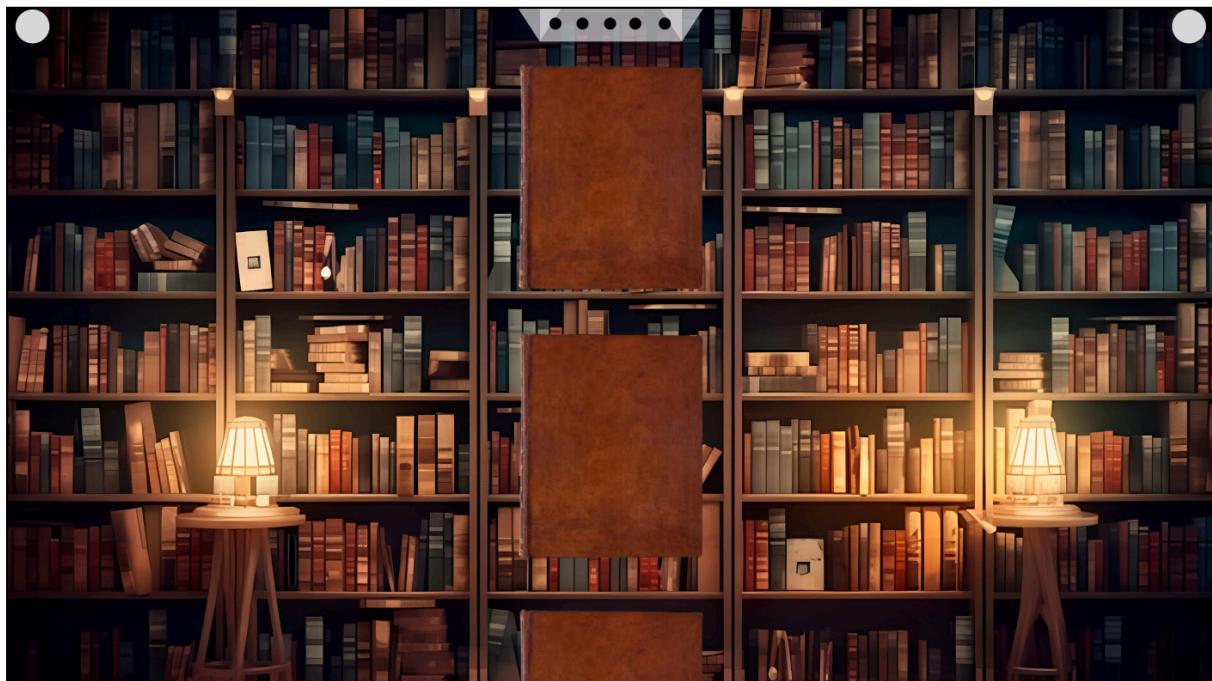


En las esquinas superiores vemos dos posibles elementos, ya pueden ser redirecciones a **GitHub**, **Portfolio**, **RRSS**, **Perfil...**

En el medio tenemos la “Navbar”, con los puntos negros siendo los elementos (**componentes a renderizar**) que he calculado que necesitaré en mi proyecto.

Me gustaría que todas las páginas sigan la **misma estructura**, por lo que más o menos serían similares entre ellas.

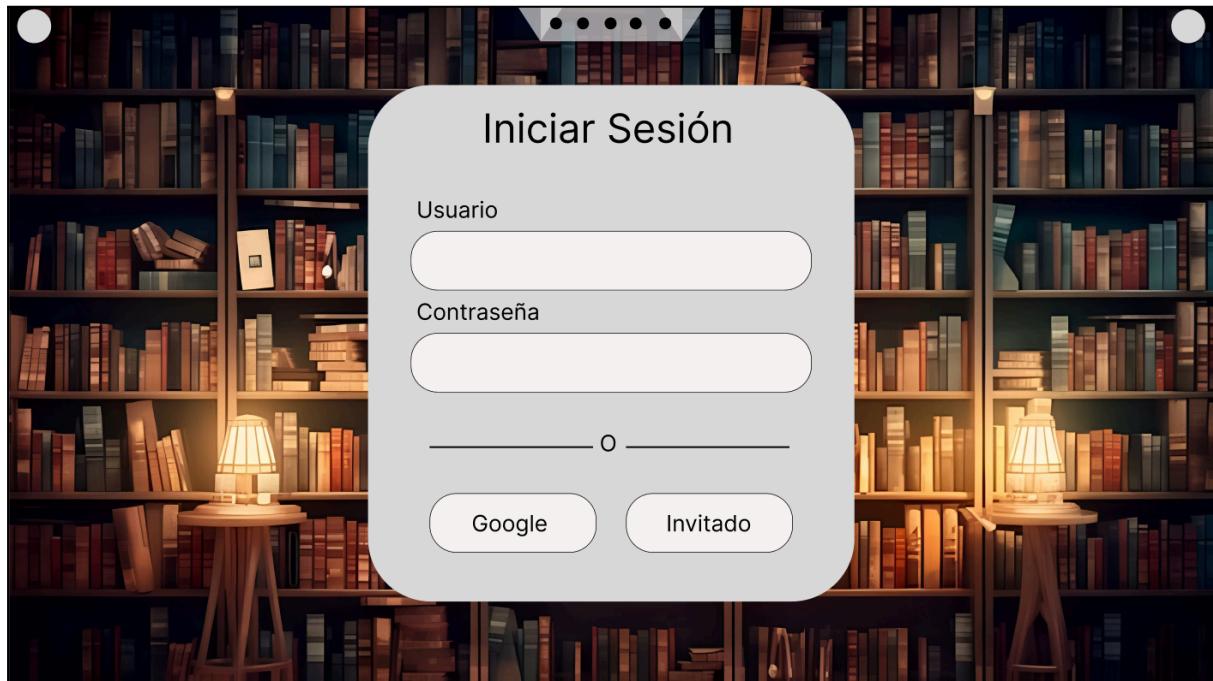
Para la web donde tenga que hacer el **select**, me gustaría una de las dos siguientes estructuras:



Finalmente opté por la **segunda opción** ya que nos permite **centrar la atención** en un sólo libro en vez de en 6 a la vez. También fue en parte por la **configuración de Svelte** y sus **estilos**, ya que tiene como un **contenedor centrado**.

Para el **LogIn** decidí elegir uno **simple y visual**, con colores pastel, ya que de esta manera quedaría más minimalista.

Mi prototipo de login es de la siguiente manera y estructura.



De esta manera, el **usuario** se **centra** sólo en lo **importante**, no en detalles insignificantes.

Los requisitos del usuario son simples.

Tiene que haber un **CRUD** con **autorización**. Un usuario **sin iniciar sesión** puede **ver los libros** que hay, el **home** y el **about**, pero **no** puede ni hacer **insert** ni **update/delete**.

Un usuario con una **sesión iniciada**, está **autorizado a todo** (incluido el **insert**) **menos update/delete**.

Usuario **admin** tiene **acceso total**, a todas las características de la aplicación. Puede eliminar, insertar, ver... **CRUD completo**.

Por otro lado, hay que realizar el **consumo** de una **API** a la **base de datos** para poder realizar el CRUD. Esa parte se hará o con **Spring Boot** o con **NodeJs y Express**.

Por último, hay que utilizar **elementos visuales** relacionados con el tema de la **magia** y los **libros**. **Tipografía** de *Harry Potter*.

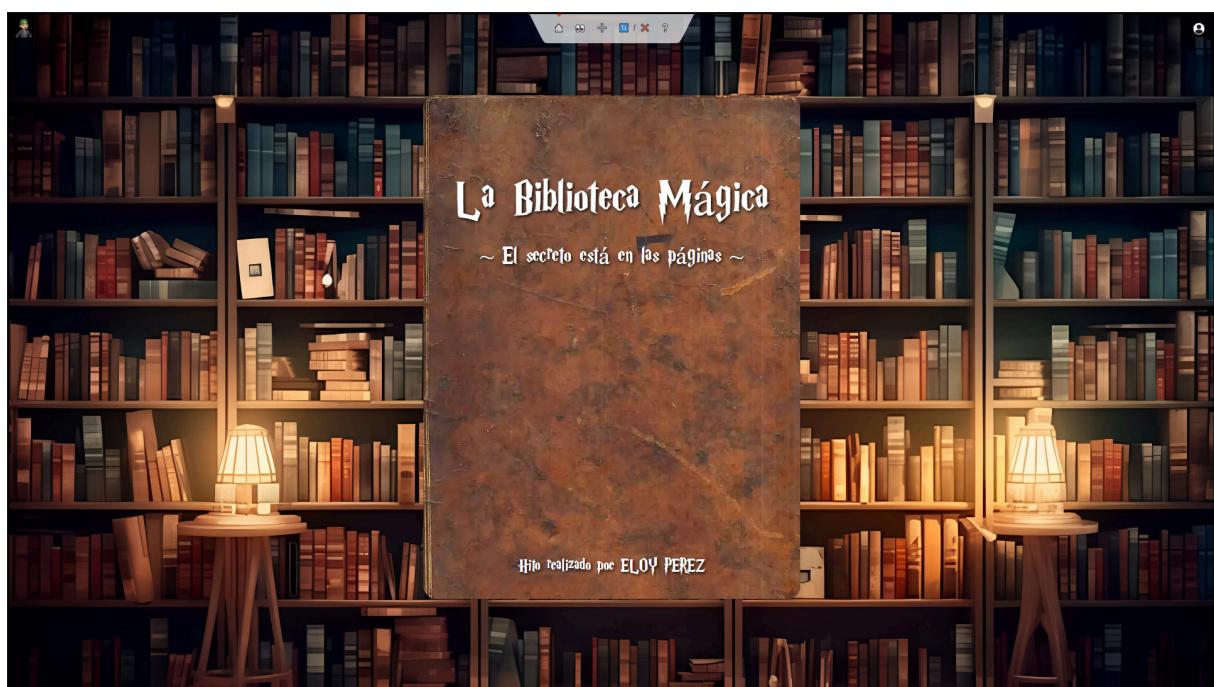
## P6 - De Design Document a Sitio Web Multipágina

Después de tener un **Design Document** bien organizado y claro, podemos ponernos a hacer el sitio web.

Para ello he usado **Svelte con Vite**, ya que es un lenguaje con muchas ventajas y características.

Después de organizar el proyecto y crear los elementos a imagen y semejanza del design document.

Mi **home** ha quedado de la siguiente manera.



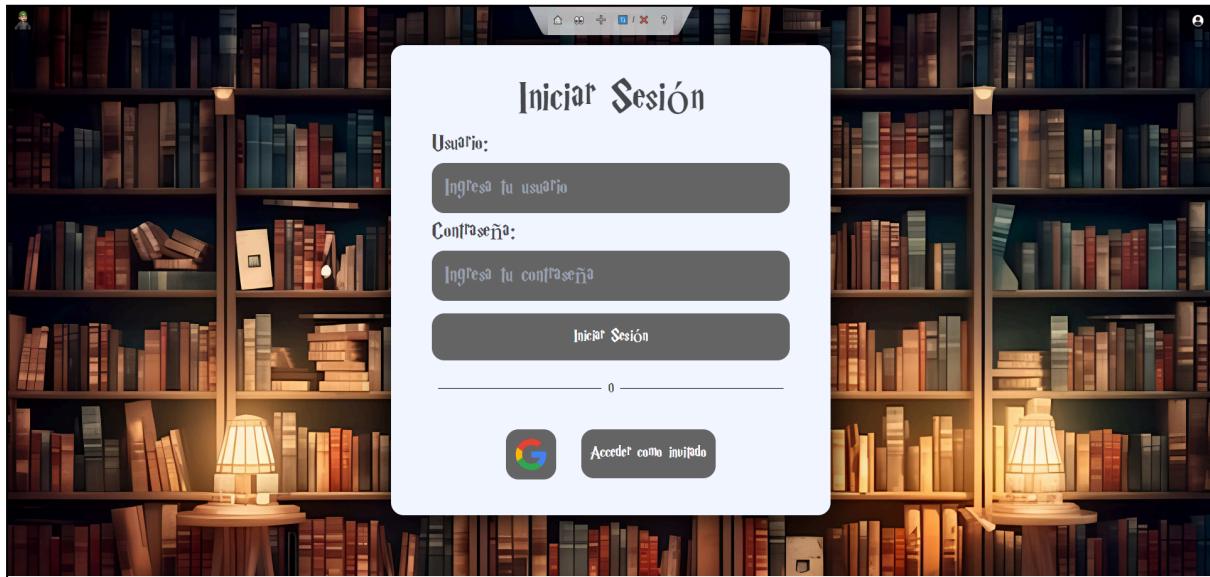
En comparación con el documento de diseño, es bastante semejante y está muy logrado. Se puede apreciar desde el inicio la temática de la web sin tener que leer ningún tipo de texto.

El componente que renderiza el **select**, luce de la siguiente manera.



Al final decidí optar por la segunda opción del documento de diseño, tanto por **configuración** como por **usabilidad y accesibilidad**.

El **LogIn** también es muy semejante al expuesto anteriormente.



Con **tipografía** de harry potter, **colores** pastel, y **botones** para iniciar sesión, con google (aunque no es funcional) y para acceder como invitado.

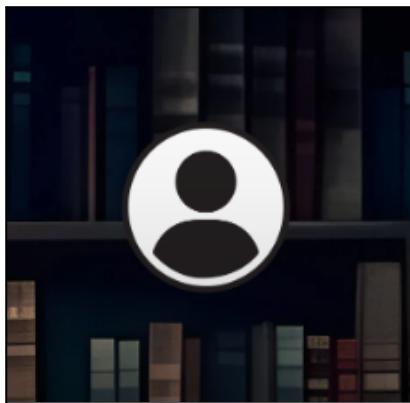
El resto de componentes siguen la estructura del componente select.

También, como último punto a tratar, los componentes superiores laterales elegidos fueron el Portfolio y el Perfil, que me llevará al LogIn.

**Portfolio:**



**Perfil:**

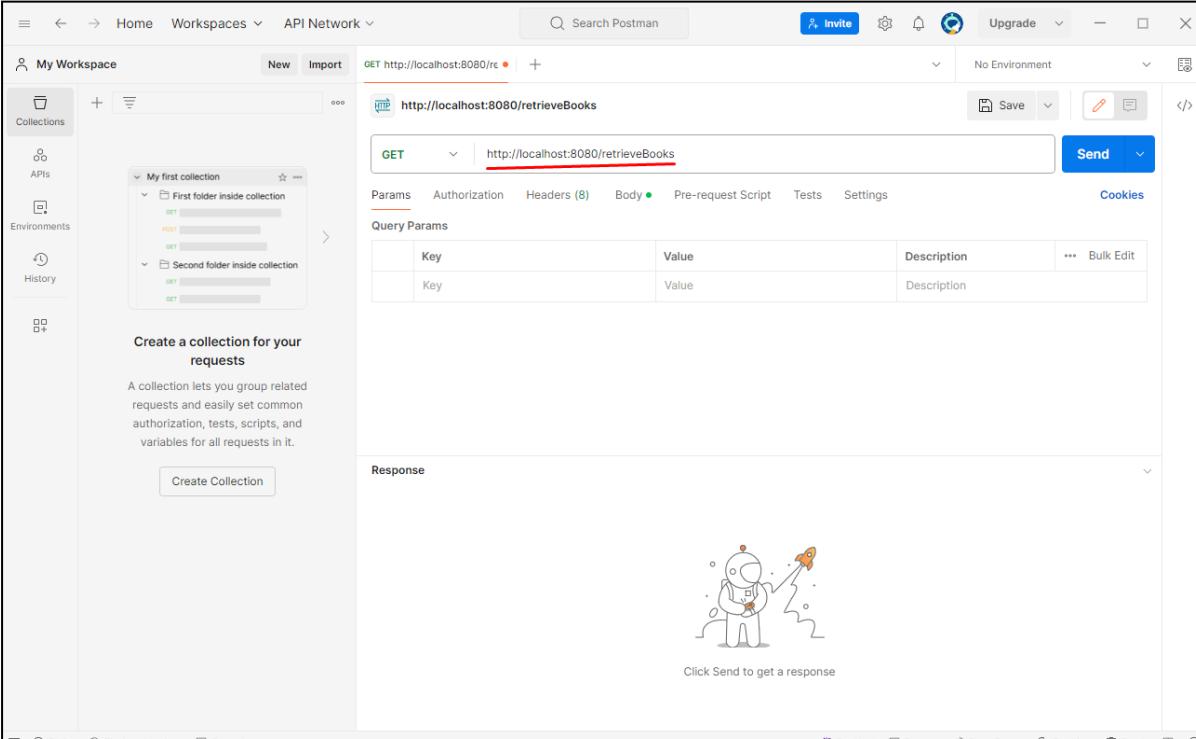


## P7 - Testing Web - Postman

Primero he decidido comprobar el funcionamiento correcto de la API que vamos a consumir.

Para ello he usado **Postman** que es una app de testing muy usada para APIs.

Así luce su menú, y subrayado en rojo vemos donde hay que colocar las rutas de las APIs.



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'My Workspace' containing 'Collections', 'APIs', 'Environments', and 'History'. A 'Create a collection for your requests' section is present. The main area displays a collection named 'My first collection' with two folders: 'First folder inside collection' and 'Second folder inside collection', each containing several requests. One request is highlighted with a red box over the URL field, which contains 'http://localhost:8080/retrieveBooks'. The request type is 'GET'. Below the request fields, there are tabs for 'Params', 'Authorization', 'Headers (8)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. A 'Query Params' table is also visible. On the right side, there's a 'Send' button, a 'Cookies' section, and a 'Response' panel featuring a cartoon character holding a rocket. At the bottom, there are various status icons and links like 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Trash', and a help icon.

Si nos acercamos un poco más, podemos ver el tipo de petición, en este caso, en verde. A eso le acompaña la ruta de la API a testear.



This is a zoomed-in view of the Postman request configuration. It shows a 'GET' request with the URL 'http://localhost:8080/retrieveBooks'. The 'Send' button is visible to the right. The background of the main workspace is visible at the top and bottom.

Si le damos a "Send" para comprobar el método GET de la API dada, vemos como efectivamente funciona.

The screenshot shows the Postman interface with a red box highlighting the 'GET' method and the URL field. The URL is set to `http://localhost:8080/retrieveBooks`. Below the URL, the 'Body' tab is selected, showing a JSON response. A second red box highlights the JSON data returned:

```
1 [  
2 {  
3     "id": 11,  
4     "name": "Camarón y la isla desierta",  
5     "author": "Camarón",  
6     "price": 32.0  
7 },  
8 {  
9     "id": 14,  
10    "name": "Francis Cofrancó",  
11    "author": "Francis",  
12    "price": 43.0  
13 },  
14 {  
15     "id": 17,  
16     "name": "Barrena el temeroso",  
17     "author": "Barrena",  
18     "price": 43.0  
19 }  
20 ]
```

Si por ejemplo queremos hacer una petición delete para borrar el que tiene ID = 11, usaremos la petición DELETE acompañado de la ruta correcta de nuestra API.

The screenshot shows the Postman interface with a red box highlighting the 'DELETE' method and the URL field. The URL is set to `http://localhost:8080/deleteBook/11`.

Donde pondremos al final "`/{id}`" para que nos lo elimine por su id. Cuando le demos a Send, si todo ha ido bien, nos dirá el siguiente mensaje.

```
1 deleted book with id 11
```

Si volvemos a hacer una comprobación, con un GET, podemos ver como se ha eliminado el libro con id = 11 correctamente.

The screenshot shows the Postman interface with the following details:

- Method:** GET (highlighted with a red box)
- URL:** http://localhost:8080/retrieveBooks (highlighted with a red box)
- Headers:** (8) (highlighted with a blue box)
- Body:** (0) (highlighted with a green box)
- Test Results:** (highlighted with a grey box)
- Pretty** (highlighted with a grey box)
- Raw**
- Preview**
- Visualize**
- JSON** (highlighted with a grey box)
- Copy** icon

The response body is displayed in JSON format:

```
1 [  
2 {  
3     "id": 14,  
4     "name": "Francis Cofrancos",  
5     "author": "Francis",  
6     "price": 43.0  
7 },  
8 {  
9     "id": 17,  
10    "name": "Barrena el temeroso",  
11    "author": "Barrena",  
12    "price": 43.0  
13 }  
14 ]
```

Ya habríamos hecho un testeo completo de la API con Postman.

Se puede seguir testeando, ya que tiene más métodos, pero para ello, tengo que tener esos métodos en mi controlador.



Después he decidido hacer el testing con **Vitest**. Vitest es un **framework** de testing de nueva generación diseñado para complementar Vite

Primero hay que instalarlo en el proyecto:

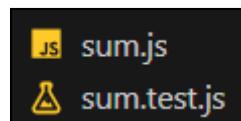
```
npm  yarn  pnpm  bun
_____
npm install -D vitest
bash
```

"vitest": "^1.2.2"

Después hay que configurar el archivo package.json creando el script:

"test": "vitest"

Tendremos que crear los archivos a testear, aquí un ejemplo básico.



Y por último hacer "npm run test" para hacer el testeo:

```
DEV v1.2.2 C:/Users/Campus FP/Desktop/Eloy/Desarrollo Web Entorno Cliente/2T/HITO2_2T_DWEC_ELOY/HITO2_
✓ sum.test.js (1)
  ✓ adds 1 + 2 to equal 3

Test Files 1 passed (1)
  Tests 1 passed (1)
  Start at 10:55:23

✓ sum.test.js (1)
  ✓ adds 1 + 2 to equal 3

Test Files 1 passed (1)
  Tests 1 passed (1)
  Start at 10:55:23
Duration 1.24s (transform 41ms, setup 0ms, collect 31ms, tests 3ms, environment 0ms, prepare 182ms)

PASS Waiting for file changes...
press h to show help, press q to quit
```

Si creamos algo más avanzado, por ejemplo el renderizado, tendríamos que hacer cosas más complicadas que una simple suma.

Yo he intentado hacerlo como he podido consultando documentación y con Copilot, la IA de Github, pero no he sido capaz de conseguir un testeo positivo.

De igual manera, nos sirve para poder ver cómo se hace un testeo.

Primero tenemos que llamar al archivo de testing de la siguiente manera.

```
<component-name>.spec.js
```

Y después meter una lógica dentro.

```
import { describe, expect, it, beforeEach, afterEach } from 'vitest';
import Page from "./+page.svelte";      Cannot find module './+page.svelte'

describe("Page", () => {
  let instance = null;      Variable 'instance' implicitly has type 'any'
                            // Create instance of the component and mount it
  beforeEach(() => {
    instance = new Page();
    instance.$$.fragment.c();
  });

  afterEach(() => {
    // Destroy/unmount instance
    instance.$destroy();      Variable 'instance' implicitly has an
                            // destroy method
  });

  it("El componente page se renderiza correctamente", () => {
    expect(instance).toBeDefined();      Variable 'instance' implicitly
    // has a defined method
  });
});
```

Sin contar los errores (que son por variables con type any), el code está más o menos bien, pero a la hora del testeo vemos que NO lo pasa por algunos otros errores.

```

FAIL src/routes/page.spec.js > Page > El componente page se renderiza correctamente
TypeError: default is not a constructor
  > src/routes/page.spec.js:9:20
    7|     beforeEach(() => {
    8|       // Create instance of the component and mount it
    9|       instance = new Page();
   10|       ^
   11|     });
  
```

---

```

FAIL src/routes/page.spec.js > Page > El componente page se renderiza correctamente
TypeError: Cannot read properties of null (reading '$destroy')
  > src/routes/page.spec.js:15:18
    13|     afterEach(() => {
    14|       // Destroy/unmount instance
    15|       instance.$destroy();
    16|     });
    17|
  
```

---

```

Test Files  1 failed | 1 passed (2)
  Tests  1 failed | 1 passed (2)
  Start at 11:16:46
  Duration 2.30s (transform 1.10s, setup 0ms, collect 1.29s, tests 11ms, environment 1ms, prepare 355ms)

FAIL Tests failed. Watching for file changes...
  press h to show help, press q to quit
  
```

Como vemos da error en ese testing, y bien en el de la suma.

```

DEV v1.2.2 C:/Users/Campus FP/Desktop/Eloy/Desarrollo Web Entorno C
  ✓ sum.test.js (1)

  ⚜ daisyUI 4.6.1
    └─ ✓ 2 themes added          https://daisyui.com/docs/themes
      ★ Star daisyUI on GitHub  https://github.com/saadeghi/daisyui
  ✓ sum.test.js (1)
  > src/routes/page.spec.js (1)
    > Page (1)
      ✘ El componente page se renderiza correctamente
        | [ beforeEach ]
        | [ afterEach ]
  
```

Habría que seguir puliendo los test para que se ajusten a lo que queremos, pero sería más o menos similar a los ejemplos dados.

# M1 Analizar el impacto de las tecnologías habituales de desarrollo web y frameworks en relación al diseño web, funcionalidad y gestión.

## Impacto de Tecnologías de Desarrollo Web:

### Diseño Web:

- **Frameworks de Front-end:** La adopción de frameworks como **React** puede simplificar el diseño al promover la creación de **componentes reutilizables**. Por ejemplo, la creación de **interfaces** de usuario interactivas en React mediante componentes como botones y formularios facilita la estructuración del diseño.
- **CSS Preprocesadores:** El uso de preprocesadores **CSS** como **SASS** impacta positivamente en el diseño al permitir la escritura de estilos más mantenibles y modulares. Por ejemplo, la capacidad de utilizar **variables** y **mixins** en SASS facilita la gestión y actualización de estilos.

### Funcionalidad:

- **Frameworks de Back-end:** La elección de un framework del lado del servidor, como **Django** en **Python**, puede tener un impacto significativo en la funcionalidad del sitio. Por ejemplo, **Django** proporciona un **ORM** (Object-Relational Mapping) integrado que facilita la interacción con **bases de datos** y simplifica el desarrollo de características complejas.
- **Libererías de JavaScript:** La adopción de **librerías** como **Axios** para realizar peticiones **HTTP** desde el lado del cliente puede mejorar la funcionalidad al gestionar eficientemente las solicitudes de datos. Por ejemplo, **Axios** facilita la implementación de comunicación **asíncrona** para cargar datos **dinámicamente** en una aplicación web.

### Gestión:

- **Sistemas de Gestión de Contenido (CMS):** La elección de un **CMS** como **WordPress** simplifica la gestión del contenido y la administración del sitio. Por ejemplo, **WordPress** proporciona una interfaz **intuitiva** para la creación y edición de contenido, facilitando la gestión para usuarios **no técnicos**.
- **Herramientas de Control de Versiones:** La implementación de **herramientas de control de versiones** como **Git** tiene un impacto positivo en la gestión del código fuente. Por ejemplo, **Git** permite un **seguimiento** preciso de los **cambios** en el código, facilitando la **colaboración** en equipo y la **gestión de versiones**.

## **Impacto de Frameworks:**

### Diseño Web:

- **Componentización:** En el caso de **Vue.js**, el componente esencial de Vue.js es su **sistema de componentes**, que facilita la creación de **interfaces modulares y reutilizables**. Por ejemplo, la creación de componentes personalizados en **Vue.js** permite una organización eficiente del diseño.

### Funcionalidad:

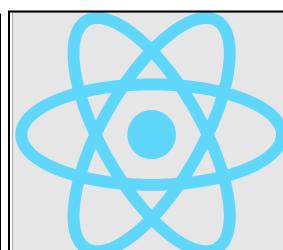
- **Desarrollo Rápido:** El uso de un framework como **Ruby on Rails** agiliza el desarrollo de funcionalidades al seguir convenciones preestablecidas y ofrecer **generadores** automáticos de código. Por ejemplo, **Rails simplifica** la creación de modelos y controladores, acelerando el desarrollo de características.

### Gestión:

- **Arquitectura MVC/MVVM:** La adopción de frameworks como **Angular**, que sigue la arquitectura **MVVM**, facilita la gestión del código al **separar** claramente la **lógica de la aplicación**. Por ejemplo, en **Angular**, los **servicios gestionan** la lógica de negocio, mientras que los **componentes se centran** en la **presentación**, lo que mejora la organización del código.



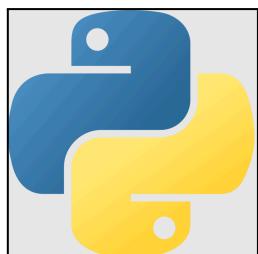
Angular



React



Ruby



Python



Django



Vue

## M2 Revisar la influencia de los motores de búsqueda en el comportamiento de los sitios web en base a su indexación y la optimización en dichos motores de búsqueda.

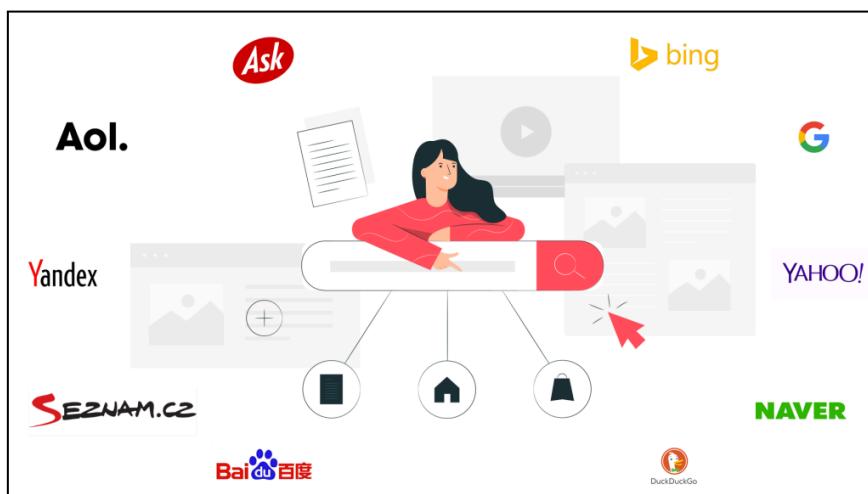
### Influencia de Motores de Búsqueda:

#### Indexación:

- **Impacto en el Comportamiento del Sitio:** La indexación por parte de motores de búsqueda, como **Google**, es crucial para determinar la visibilidad del sitio web. Sitios **bien indexados** tienen **mayores posibilidades de aparecer en los resultados de búsqueda**, afectando directamente el tráfico y la visibilidad.
- **Optimización para la Indexación:** Estrategias de **SEO** (Search Engine Optimization) son esenciales para **mejorar la indexación**. Elementos como **metaetiquetas, sitemaps** y contenido de **calidad** influyen positivamente en cómo los motores de búsqueda interpretan y clasifican un sitio.

#### Optimización en Motores de Búsqueda:

- **Comportamiento y Posicionamiento:** La **optimización** para motores de búsqueda impacta el comportamiento del sitio en términos de posición en los resultados de búsqueda. Estrategias éticas y efectivas de SEO ayudan a mejorar el posicionamiento, aumentando la probabilidad de que los usuarios encuentren el sitio.
- **Factores Clave de SEO:** La **calidad** del contenido, la **estructura** del sitio, la **velocidad** de **carga** y la relevancia de las palabras clave son factores que afectan la optimización en motores de búsqueda. Mantener estas prácticas contribuye al éxito continuo del sitio en el entorno digital.



## **Análisis Crítico:**

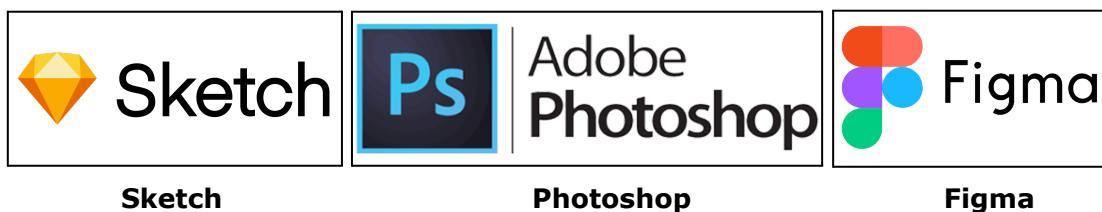
La influencia de los motores de búsqueda es innegable en el entorno web actual. La **indexación precisa** y la **optimización efectiva** son elementos **cruciales** para el **éxito** de un sitio. Sin embargo, es **esencial** adoptar un **enfoque equilibrado y ético** en el **SEO**, evitando prácticas que busquen **manipular** resultados.

Es necesario reconocer que los **algoritmos** de los motores de búsqueda **evolucionan**, y lo que funciona hoy puede **necesitar ajustes** en el futuro. El análisis crítico implica **adaptarse** a estos **cambios** y **mantener** una estrategia de SEO que priorice la **auténticidad** y la **utilidad** para los usuarios sobre tácticas puramente orientadas a la **optimización**.

## M3 - Analizar el conjunto de herramientas y técnicas disponibles para el diseño y desarrollo de un sitio web personalizado.

### **Herramientas para el Diseño:**

**Editores de Gráficos:** Se pueden usar herramientas como **Adobe Photoshop**, **Sketch** o **Figma** para crear elementos gráficos personalizados, como imágenes de fondo, iconos o incluso logotipos.



**Prototipado y Diseño de Interfaz:** Herramientas como **Adobe XD**, **InVision** o **Figma** para crear prototipos interactivos que representen la estructura y la navegación del sitio web.

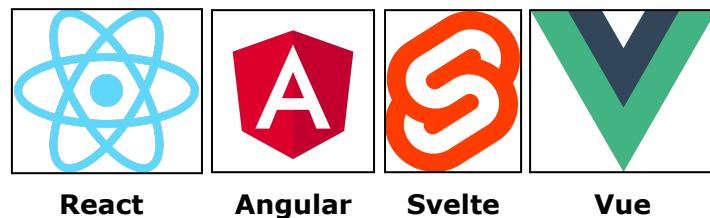


**Editores de Texto Avanzados:** Selecciona editores como **Visual Studio Code**, **Sublime Text** o **Atom** para escribir y editar código *HTML*, *CSS* y *JavaScript* de manera eficiente.



### **Técnicas para el Desarrollo:**

**Desarrollo Front-end:** Utiliza **frameworks** como **React**, **Angular**, **Svelte** o **Vue.js** para desarrollar interfaces de usuario interactivas y reactivas. También se pueden implementar preprocesadores CSS como **Sass** o **Less** para facilitar la escritura y gestión de estilos.

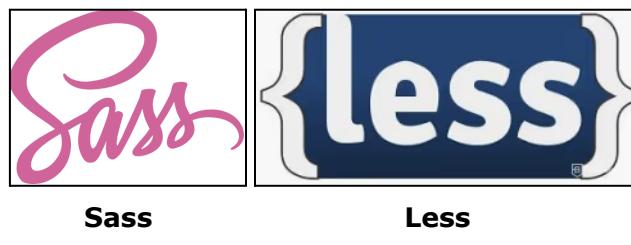


React

Angular

Svelte

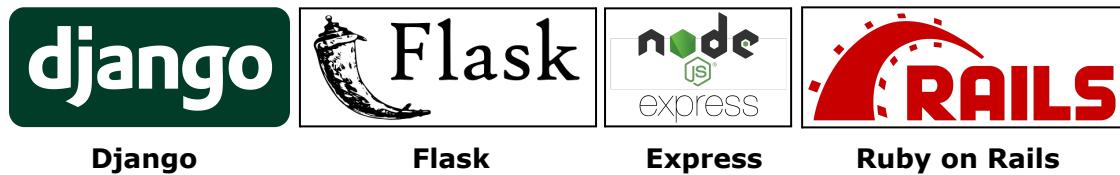
Vue



Sass

Less

**Desarrollo Back-end:** Emplea **frameworks** como **Django**, **Flask** (*Python*), **Express** (*Node.js*) o **Ruby on Rails** para construir la lógica del servidor y gestionar las interacciones con la base de datos.



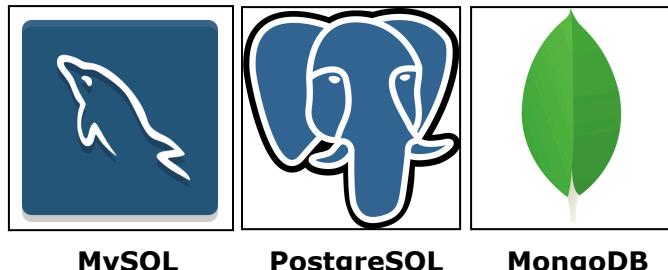
Django

Flask

Express

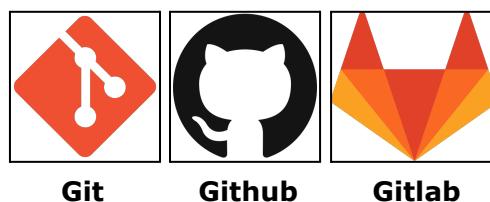
Ruby on Rails

**Bases de Datos:** Selecciona *bases de datos SQL* (**MySQL**, **PostgreSQL**) o *NoSQL* (**MongoDB**) según los requisitos específicos del proyecto. También se pueden utilizar herramientas de administración de bases de datos, como **MySQL Workbench** o **MongoDB Compass**, para gestionar datos.

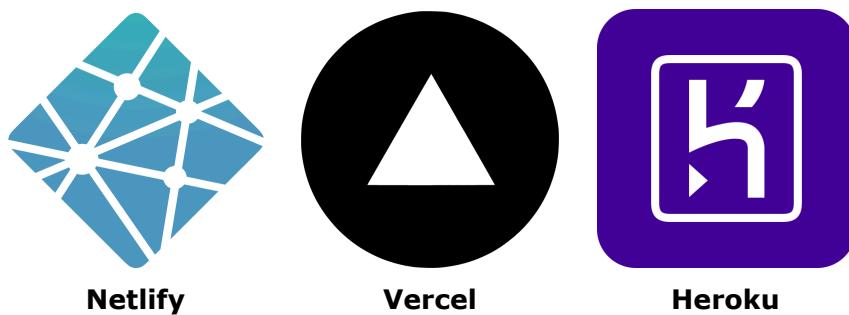


#### **Control de Versiones:**

- Implementa sistemas como **Git** para el control de versiones y **GitHub** o **GitLab** para colaboración y alojamiento remoto.



**Despliegue y Gestión de Servidores:** Podemos desplegar el sitio web en plataformas como **Netlify** (*gratis*), **Vercel** (*gratis*), **Heroku** (*de pago*) o utilizar servicios en la **nube** como **AWS** o **Azure**. Configuraremos y gestionaremos los servidores utilizando herramientas como **Nginx** o **Apache** (*por ejemplo*).



## M4 Justifica las decisiones de implementación del sitio web respecto al design document.

### 1. Diseño de la Interfaz y Componentes Visuales:

La elección de una **estructura uniforme** en todas las páginas con una **barra de navegación** en la parte superior junto con los dos elementos laterales, asegura coherencia y, sobre todo, *facilita la navegación* del usuario.

La inclusión de la **tipografía** original de Harry Potter y **colores pastel oscuros** crea una *experiencia visual inmersiva* y alineada con la temática de "The Magic Library".

Se utilizará la **tipografía** de **Harry Potter** (ej. "Wizard's Brew") para mantener la coherencia con el tema mágico.

Los botones con efecto **hover** en formularios y elementos interactivos contribuyen a la experiencia visual y a la intuitividad del usuario.

### 2. Componente Interactivo Adicional - Cursor custom Harry Potter:

La adición del cursor personalizado referente a Harry Potter agrega un toque distintivo y mágico a la experiencia del usuario. Sin embargo, es importante evaluar su viabilidad técnica para evitar posibles **problemas de rendimiento**.

Tendremos que realizar **pruebas exhaustivas** para garantizar que no afecte negativamente al **rendimiento** del sitio.

Asegurarse de que la animación sea **sutil** y no **distraiga** demasiado al usuario, considerándose una **animación de fondo**.



Perdón por la calidad, pero no se puede capturar el puntero en una captura de pantalla

### **3. Requisitos Funcionales - CRUD con Autorización:**

La implementación de un **CRUD** completo con **autorización** para insertar, actualizar y eliminar datos garantiza un manejo efectivo de la información en la “biblioteca”. La autorización ayuda a mantener la **integridad** de los datos y la **seguridad** del sitio.

Utilizar un sistema de gestión de usuarios con **roles** para controlar el acceso a las funciones de CRUD.

Implementar **formularios seguros** y **validaciones** para prevenir posibles ataques.

### **4. Requisitos del Usuario - Navbar Visual y Contenido Intuitivo:**

La creación de una **barra de navegación visual** con emoticonos mejora la **usabilidad** y la navegación **intuitiva**. La inclusión de **imágenes** relacionadas con el tema mágico contribuye a un entorno **atractivo** y **fácil** de entender.

Utilizar **emoticonos** relevantes en la barra de navegación para representar de manera visual cada sección.

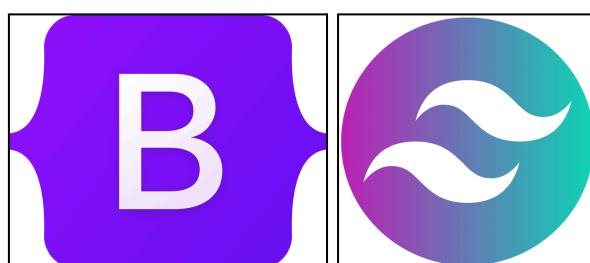
Optimizar las imágenes para una **carga rápida** y una experiencia de usuario fluida.

### **5. Adaptabilidad y Responsividad:**

La decisión de utilizar **unidades relativas (%)** y estilos preestablecidos, como **Bootstrap** o **Tailwind**, asegura que el sitio sea **responsive** y **adaptable** a diferentes dispositivos.

Realizar **pruebas** en dispositivos con diversos tamaños de pantalla para garantizar que la interfaz se ajuste adecuadamente.

Emplear **media queries** y técnicas de diseño responsivo para abordar situaciones específicas.



Bootstrap

Tailwind

## M5 Analiza el proceso QA y review cómo fue implementado durante las etapas de diseño y desarrollo del sitio web.

Primero que nada, el Aseguramiento de Calidad (QA) es esencial para garantizar que la web cumpla con los estándares de calidad definidos.

La calidad del sitio web tiene un impacto directo en la experiencia del usuario y la satisfacción del cliente.

### 1. Proceso de QA en el Diseño:

- Actividades de QA durante el Diseño:
  - Llevé a cabo revisiones de **prototipos** y **wireframes** para asegurar coherencia con los requisitos del cliente.
  - También realicé pruebas de **usabilidad** y **accesibilidad** para validar el diseño respecto a estándares web.
- Herramientas utilizadas:
  - Utilicé **Figma** y **Paint** para la creación de prototipos.
  - Se puede validar la accesibilidad mediante el uso de herramientas como **Axe Accessibility Checker**.



Figma



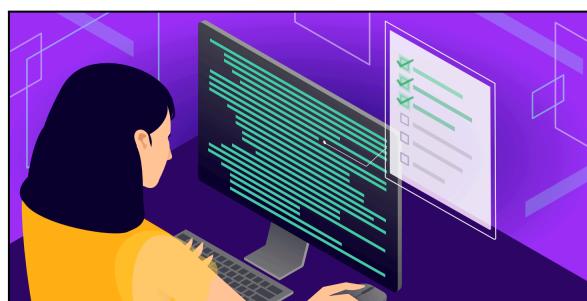
Paint



AAC

### 2. Proceso de QA en el Desarrollo:

- Actividades de QA durante el Desarrollo:
  - Se implementan **pruebas de funcionalidad** para asegurar que todas las características se implementan correctamente.
  - Se realizan **pruebas de rendimiento** para evaluar y optimizar la velocidad de carga del sitio.
  - Se llevan a cabo **revisiones de código** para garantizar calidad y coherencia en la base de código.



Revisión de código

### 3. Herramientas y Técnicas de QA:

- Herramientas Específicas:
  - Se pueden emplear **herramientas de seguridad web** para proteger las comunicaciones y prevenir **vulnerabilidades** o **hackeos**.
  - Se implementan **análisis estáticos** y **dinámicos** de código para **fortalecer la seguridad**.
- Técnicas con posibilidad de implementación:
  - Podríamos usar la **técnica de pruebas de penetración** para identificar **posibles brechas** de seguridad.

### 4. Desafíos y Soluciones en el Proceso de QA:

- Desafíos Enfrentados: En la **optimización** del rendimiento y la **compatibilidad** con navegadores específicos (problemas con el CORS en Google Chrome).
- Soluciones Implementadas: Técnicas de **optimización de imágenes** y **recursos** para abordar problemas de rendimiento (convertir a webp para que pesen menos las imágenes...).  
Pruebas en diferentes navegadores para **garantizar compatibilidad**.

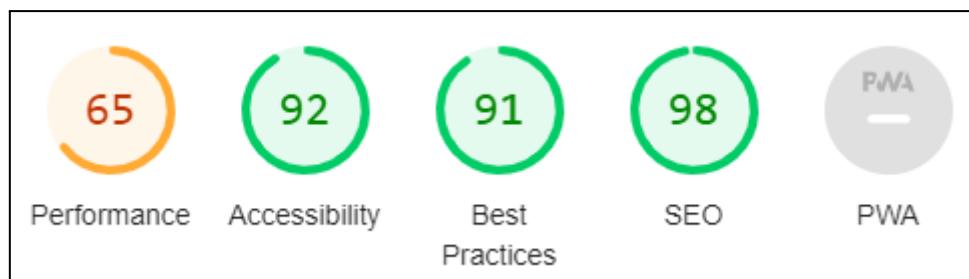
### 6. Conclusiones:

La importancia de pruebas continuas y revisiones regulares para mantener la calidad. Se sugiere explorar herramientas de automatización para agilizar procesos de QA.



Configuración Web

A continuación vemos una comprobación de QA implementada automáticamente con Google Chrome: [Lighthouse](#)



Vemos como en la parte de Performance flojea la web.  
Habrá que optimizarla para que eso no ocurra. Para ello podemos aplicar distintos métodos:

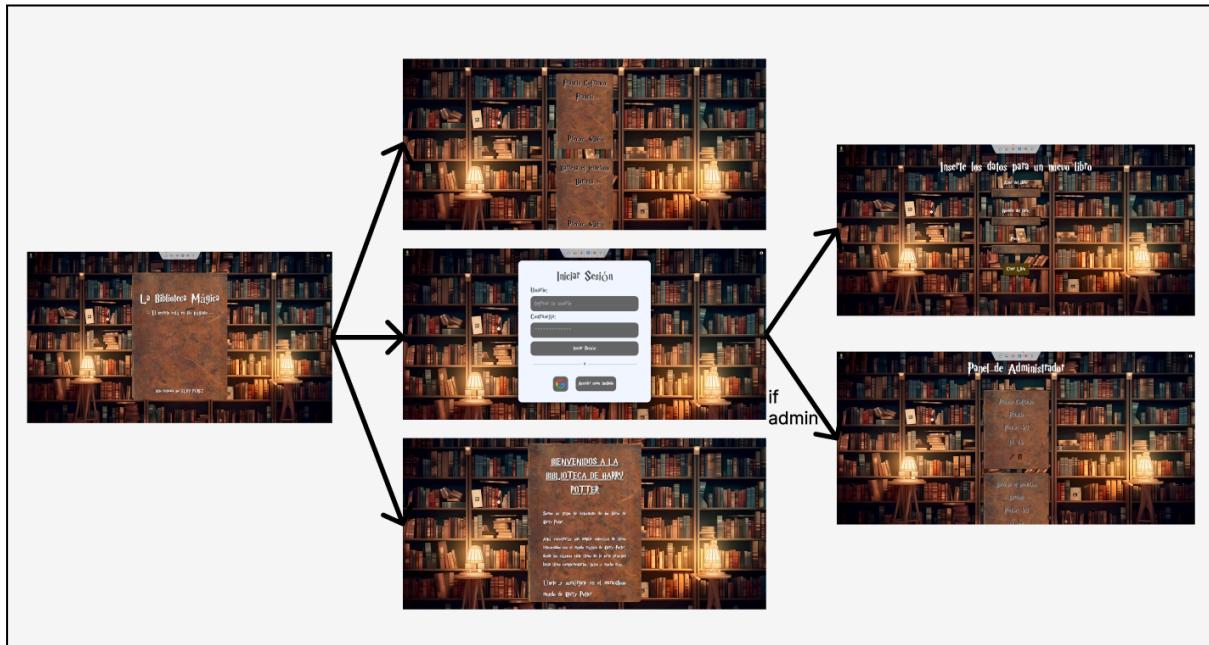
- **Tamaño de los archivos multimedia:** Cambiando todas las extensiones posibles a webp, así reducimos el tamaño del archivo y está optimizado para web.
- **Reduciendo elementos de JS:** Si reducimos elementos como alerts() o console.log(), mejoraremos el rendimiento de la web.
- **Reciclando estilos:** No poner dos clases duplicadas con los mismos estilos y en su lugar usar sólo una

Si aplicamos estos métodos, el rendimiento de la web mejorar notablemente.

Otro método de saber qué podemos hacer para optimizarla, nos lo dice el propio Lighthouse:

DIAGNOSTICS	
▲	Largest Contentful Paint element — 16,650 ms
▲	Minimize main-thread work — 2.5 s
▲	Enable text compression — Potential savings of 886 KiB
▲	Minify JavaScript — Potential savings of 906 KiB
▲	Properly size images — Potential savings of 260 KiB
▲	Reduce unused JavaScript — Potential savings of 94 KiB
■	Ensure text remains visible during webfont load
■	Page prevented back/forward cache restoration — 1 failure reason

**Happy Path:** Es el **escenario de uso ideal** de un software sin condiciones de excepción, en el que **nada funciona mal**, nada fuera de lo normal ocurre y todo se desarrolla de forma **rápida y directa** hasta lograr el objetivo deseado por el usuario.



Ese sería el **Happy Path** de mi web, el camino básico a seguir sin ningún tipo de excepción. Es la navegación por todas las páginas posibles, incluyendo la de administrador.

## D1 Justificar las tecnologías, servicios, herramientas y software elegidos para la realización del sitio web.

La parte de front está hecha con **Svelte** usando **Vite**. Usé Vite ya que sirve para desarrollar más rápido la estructura del proyecto. Aparte, tiene un excelente **soporte** para Svelte, por lo que se puede configurar y buscar dudas en internet. También usa **módulos** de **ECMAScript**, por lo que sólo carga lo necesario para el proyecto.

Está integrado con Svelte, **SvelteKit**, que es un marco de web completo basado en Svelte beneficiado de la **velocidad** que proporciona Vite.

He usado Svelte por múltiples razones. La primera es que **compila en tiempo de ejecución**, por lo que puedo estar trabajando con dos monitores y ver en uno el código y en otro lo que voy desarrollando.

La segunda razón es porque el código generado es **puro JavaScript** y **no necesita bibliotecas adicionales** durante la ejecución.

Svelte también cuenta con una **sintaxis** bastante **sencilla**, por lo que no resulta difícil comprender su código. Como último punto positivo pondría que tiene una **comunidad** bastante **activa** y puedes encontrar la solución a las dudas en internet si buscamos un poco.

En la parte del “back” he usado **Spring Boot** con **IntelliJ** haciendo una conexión a **MySQL** usando **XAMPP**. Suena confuso pero realmente es sencillo. Primero usé **XAMPP** lanzando **MySQL** y trabajé sobre la base de datos “test”. Me conecté con **Java (Spring Boot)** a MySQL usando **IntelliJ** como **entorno**. Ahí es donde tenía la gestión de mi base de datos y mi **API a consumir en el front**.

La ruta del back sería “*localhost:8080*” y la del front es “*localhost:5173*”. Desde el front **me conecto** a la ruta del back **como** una **API**. Y gracias a las configuraciones que tengo con Spring Boot puedo hacer **insert**, **select**, **delete**...

Por ejemplo, para hacer el **select**, en mi caso, me conecto como API a “*localhost:8080/retrieveBook*”. La consumo con un **async await fetch** y muestro sus datos en pantalla. Para hacer el **insert** puse a “*.../createBook*”, y así con todo.

También se podría haber hecho otra cosa, igual sin tanto lío. Hacer el **back con node y express**. Me conectaría a la base de datos con node y haría lo correspondiente para hacer las funcionalidades.

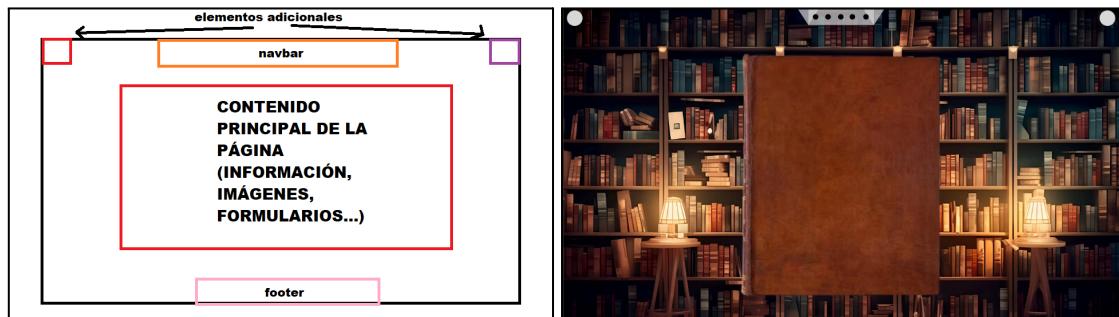
En cuanto a las versiones, se usa ES6 o superior, la versión 2.4.5 del parent de Spring Boot, la 5.0.3 de Vite y la 4.2.7 de Svelte.

```
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>2.4.5</version>
```

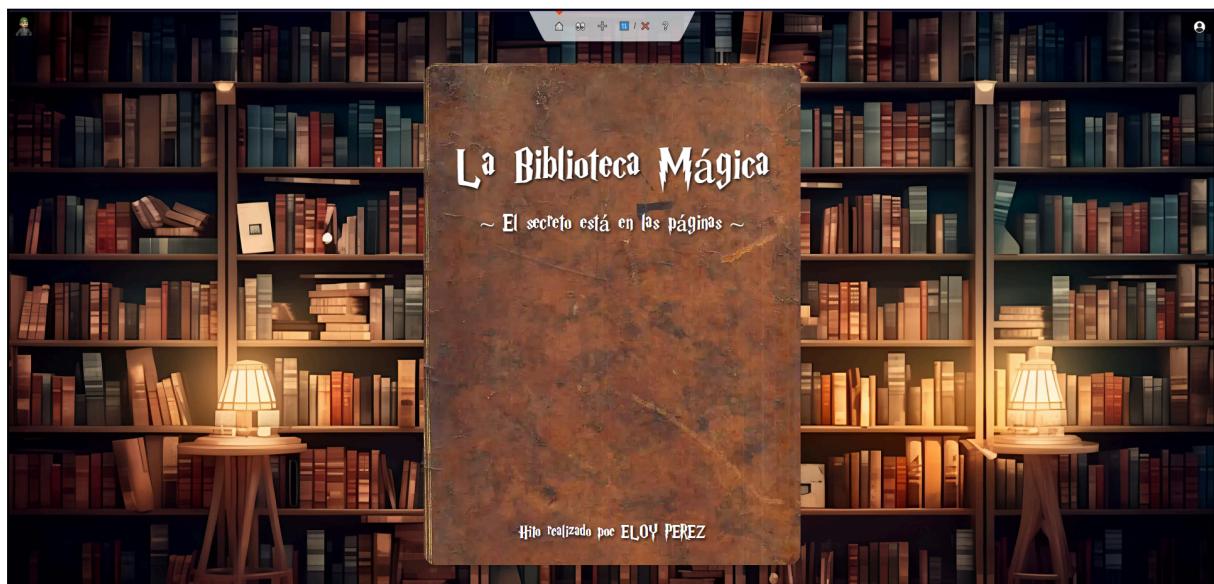
"vite": "^5.0.3"
"svelte": "^4.2.7",

## D2 Evalúa el diseño y el proceso de desarrollo del sitio web respecto al design document incluyendo algunos desafíos técnicos a los que te has enfrentado

Finalmente, la main page comparada al documento de diseño es tal que así:



Distribuciones de los prototipos



Main page final

Vemos como es muy similar al documento de diseño. Los elementos están relacionados con el tema tratado y son coherentes.

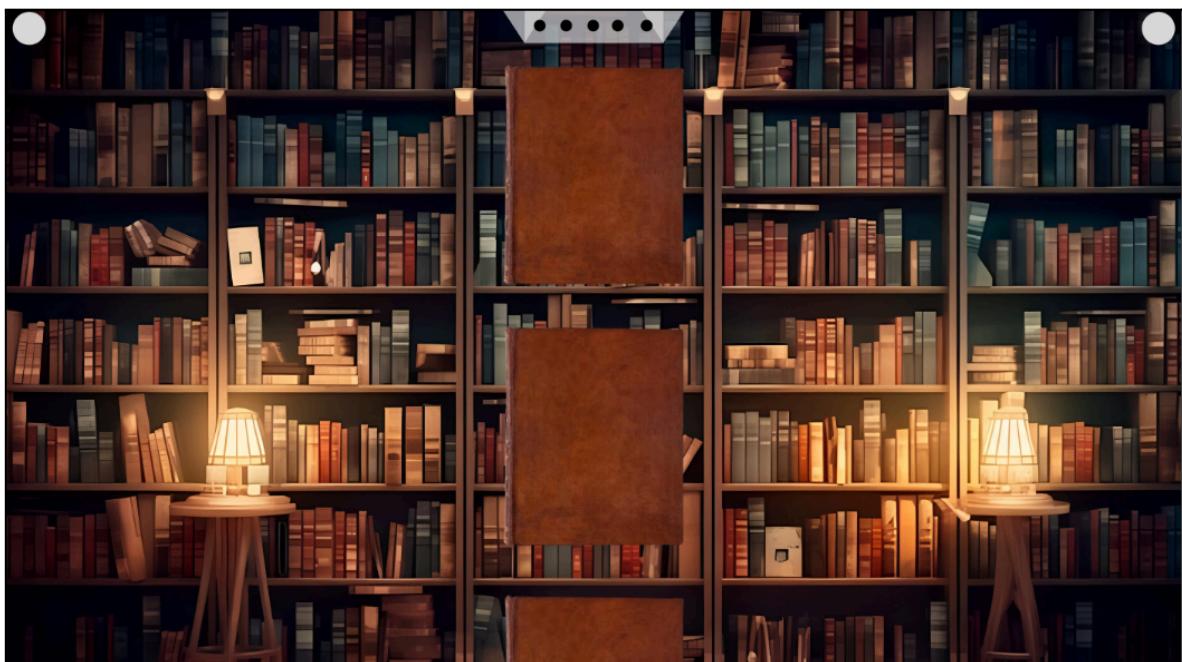
Estructura muy similar, contando con navbar, elementos laterales, y contenido multimedia relacionado con el tema.

He tenido complicaciones a la hora de la distribución y las alineaciones, las cuales he conseguido solventar con estilos css.

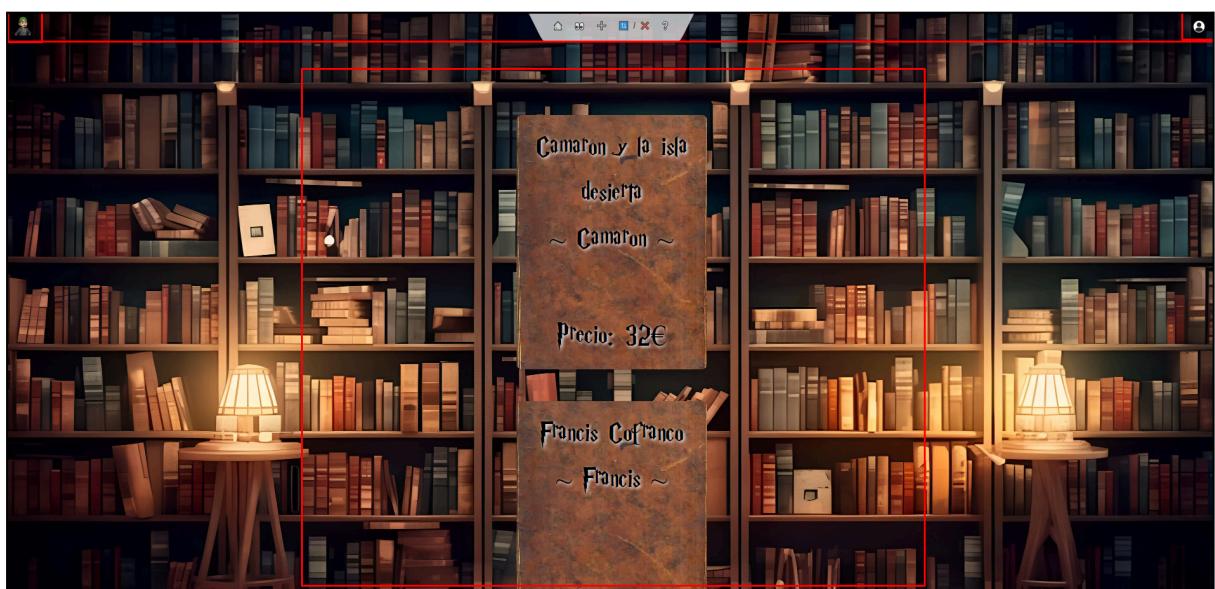
Una de las peores complicaciones a la hora de desarrollar todo el proyecto han sido las importaciones de imágenes o tipografía. Principalmente porque no me encontraba el elemento de la manera básica. Tuve que recurrir a un import en JavaScript para luego usarlo como variable.

```
import fondo from '$lib/images/fondoLibro.webp';
import cursor1 from '$lib/images/cursor1img.png';
```

También contamos con la página de select por ejemplo, que sigue la misma dinámica. Con la navbar y el contenido.

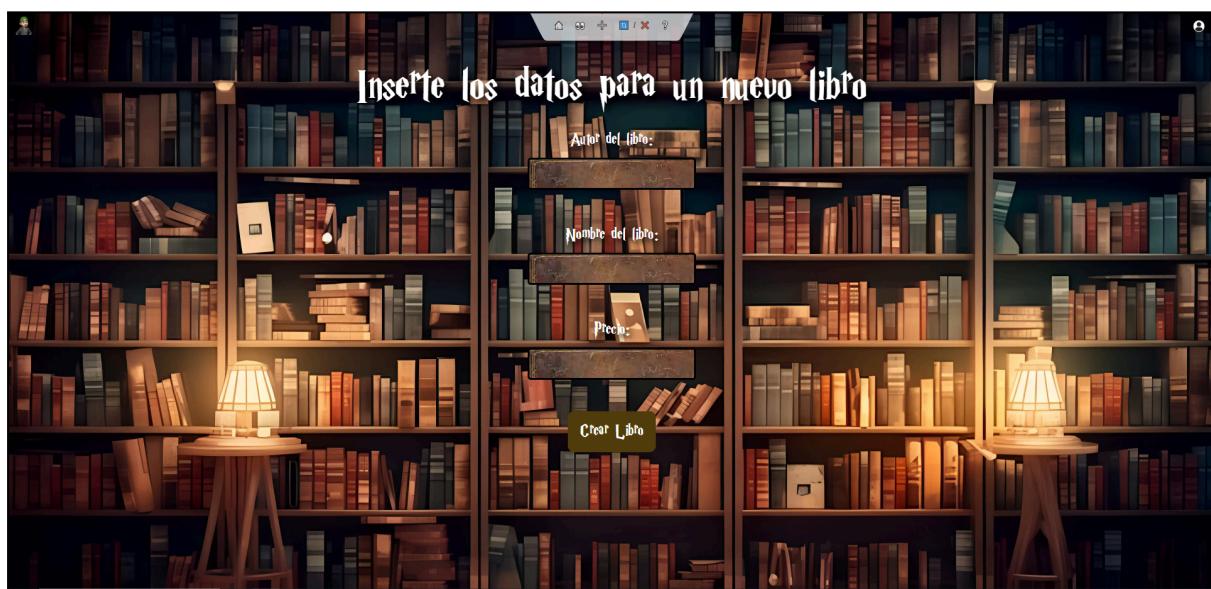


Select Prototype

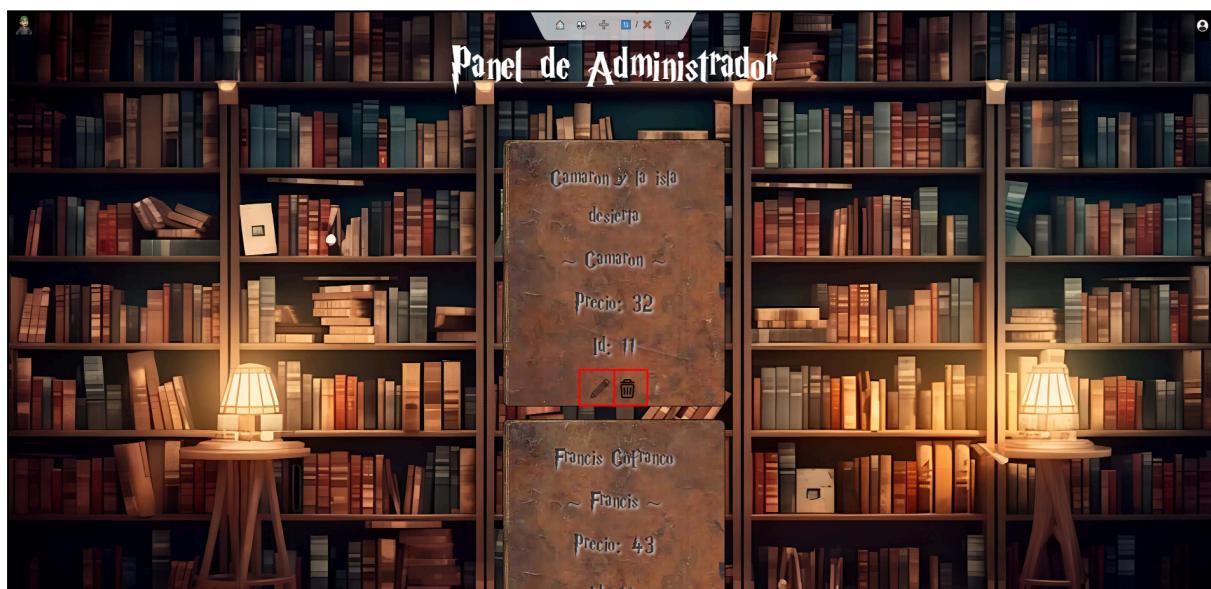


Select page final

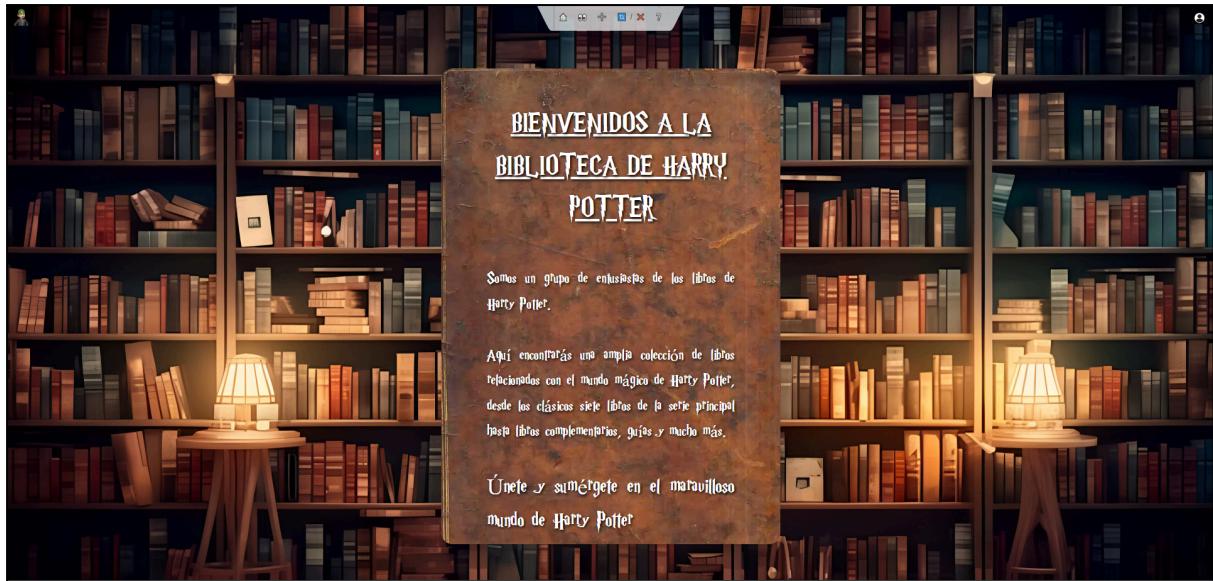
A continuación muestro las páginas de insert, update delete y about ya que siguen la estructura.



Insert page



Update/Delete page

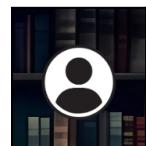


About page

Esas serían las páginas que siguen la misma estructura.

Finalmente no he tenido mucho problema ya que en el momento que tenía una bien realizada con sus estilos y su distribución, podría copiar y pegar la carpeta para posteriormente añadirla a la ruta.

Pero si ha habido una página, no muy oculta, que tiene similar distribución pero no igual. Ese es el apartado de login, al que se accede dándole al icono del perfil en la parte superior izquierda.

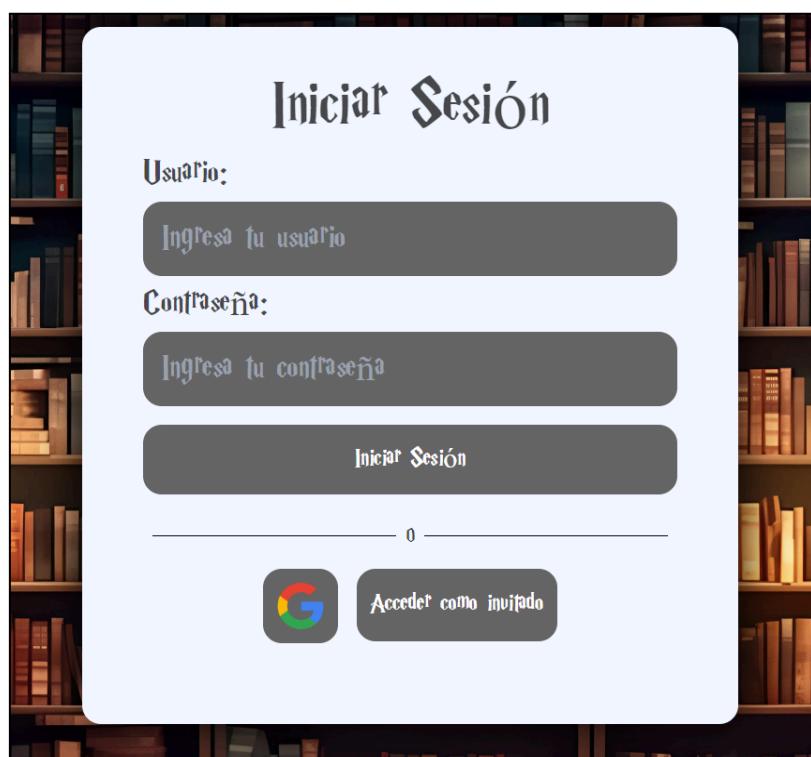


Nos llevará a la siguiente web:



LogIn page

Un login atractivo, con botones muy visuales y claros. Se podría interpretar que sigue la misma estructura, que en parte sí, pero realmente es un conjunto de <div> y <section> dotados de estilos para lograr ese aspecto llamativo.



LogIn

Lo más complicado de todo esto, han sido las **funcionalidades**. También el hecho de alinear todo dándole los *estilos, fondos, bordes, sombreados* y que estuvieran en la *posición deseada*.

Por el momento el **inicio de sesión con Google** no está disponible, pero me gustaría mejorar este proyecto en un futuro y poder **implementarlo** tanto este, como otras funciones.

Mi intención con el LogIn with Google era hacerlo con **OAuth2**, pero debido a algunos problemas con el back, el desconocimiento y las prisas, no veo viable implementarlo, pero no descarto que con más tranquilidad poder **mejorar** el

para implementar tanto esta como otras funcionalidades que mejoran la web.

También tuve bastantes problemas con el consumo de la API, ya que hay muchas maneras de consumirla.. Finalmente acabé realizandolo con un **async await fetch** que me devolvía los datos perfectamente, y además aplicaba la **asincronía** a la web.

Aquí hay una foto del código de cómo hago el consumo.

```
import { onMount } from 'svelte';
let catFact = [];

onMount(async () => {
    const response = await fetch('/retrieveBooks');
    const data = await response.json();
    catFact = data;
    console.log(catFact);
});
```

Consumo de la API

Donde “catfact” es donde almaceno el **consumo**. La variable *response consume la API*, después *data* la **convierte response en JSON**. Por último se asigna a **catFact = data** para pasarselo el **JSON legible**.

## **D3 Evalúa los resultados del Test Plan y el éxito completo del sitio web con recomendaciones para su mejora**

El test plan no ha sido exitoso. Como pudimos ver en el p7 a la hora de comprobar si nos renderizaba el componente, salían fallos. Pero esos fallos no tienen que ver con el componente, si no con la forma de testeo.

Para mejorarlo tendríamos que seguir documentándonos y seguir probando hasta conseguir el desarrollo exitoso del test.

Al igual que hemos intentado el test usando vitest para el main component, podríamos hacerlo para comprobar que se renderizan todos los componentes existentes de nuestra app web.

Podríamos comprobar que se renderiza el select, el login, el insert...

Para mejorar la aplicación haría varias cosas:

- **Mejorar el aspecto visual:** Soy consciente de que el aspecto visual no está mal, pero podría ser mejor.
- **LogIn Google:** Me gustaría hacer una futura implementación del LogIn con google.
- **Probar con otros Frameworks:** Me gustaría probar a realizar la aplicación con otros frameworks como React, Vue, Angular para aprender a ser multilenguaje.
- **Probar con otro back:** Al igual que me gustaría probar con otros frameworks, también me gustaría probar con otro back. Como ya mencioné anteriormente en el documento, se puede realizar el back con nodejs.
- **Botón de LogOut:** Actualmente NO cuenta con esta funcionalidad, hay que borrar la cookie creada manualmente, pero se podría mejorar bastante la aplicación incluyendo un botón que la borre con pulsarlo y después tener que volver a iniciar sesión.

## WebGrafía

1. **Cloudflare - What is DNS?** Cloudflare Learning Center - Consultado en  
<https://www.cloudflare.com/es-es/learning/dns/what-is-dns/>
2. **Cloudflare. (s.f.). DNS Server Types.** Cloudflare Learning Center - Consultado en  
<https://www.cloudflare.com/es-es/learning/dns/dns-server-types/#:~:text=%C2%BFCu%C3%A1les%20son%20los%20tipos%20diferentes,y%20servidores%20de%20nombres%20autoritativos>
3. **Medium - Svelt con OAuth2** - Consultado en  
<https://medium.com/@mateuszpiorowski/oauth2-is-so-complicated-or-90-lines-of-code-with-svelte-ab0f5d80d659>
4. **JustInMind - Inspiring login pages** - Consultado en  
<https://www.justinmind.com/blog/20-inspiring-website-login-form-pages/>
5. **Cómo poner el cursor custom en HTML** -  
<https://www.youtube.com/watch?v=OKvvjXu7WE8>
6. **Vitest - Get Started** - <https://vitest.dev/guide/>
7. **Testing Svelte Vitest - How to do it** -  
<https://blog.logrocket.com/testing-svelte-app-vitest/>