

Ace Books

Alejandro Leal Bermúdez

IES Vega de Mijas

2018/2019

Índice

1. Introducción.	1
2. Descripción del proyecto.....	1
2.1 Objetivos.	1
2.2 Requisitos mínimos y restricciones.	1
3.Métodos y herramientas de desarrollo.	1
3.1 Hardware.....	1
3.2 Software.	1
4. Diseño.....	2
4.1 Diagrama entidad-relación y tablas normalizadas.	2
4.2 Estructura de la aplicación.	2
4.3 Flujo de control de cada una de las páginas PHP que componen el proyecto.	6
4.4. Interfaz.	10
5. Programación	14
6. Conclusiones.....	17
7. Mejoras.	17
8. Bibliografía.	18

1. Introducción.

Actualmente hay distintas páginas web que intentan ofrecer un servicio de lectura de documentos, sin embargo, todas estas se quedan cortas, sea por no soportar varios tipos de documentos, carecer de la posibilidad de organizar tus documentos e incluso de no tener un buen lector de documentos.

Ace Books te da la capacidad de leer tus archivos pdf, organizar estos en distintos directorios e incluso interactuar con ellos.

2. Descripción del proyecto.

2.1 Objetivos.

Los objetivos del proyecto son los siguiente:

- Ofrecer al usuario las herramientas necesarias para organizar sus documentos en colecciones.
- Poder personalizar los documentos y colecciones (nombre, descripción y portada).
- Un lector de documentos.
- Gestionar tu información (nombre usuario, imagen de perfil, contraseña e idioma de la web).

2.2 Requisitos mínimos y restricciones.

Los usuarios tendrán que registrarse para usar los distintos servicios que son ofrecidos. Se necesitará acceso a internet y un navegador moderno.

3. Métodos y herramientas de desarrollo.

3.1 Hardware.

Para el desarrollo del proyecto he usado mi PC que tiene los siguientes componentes:

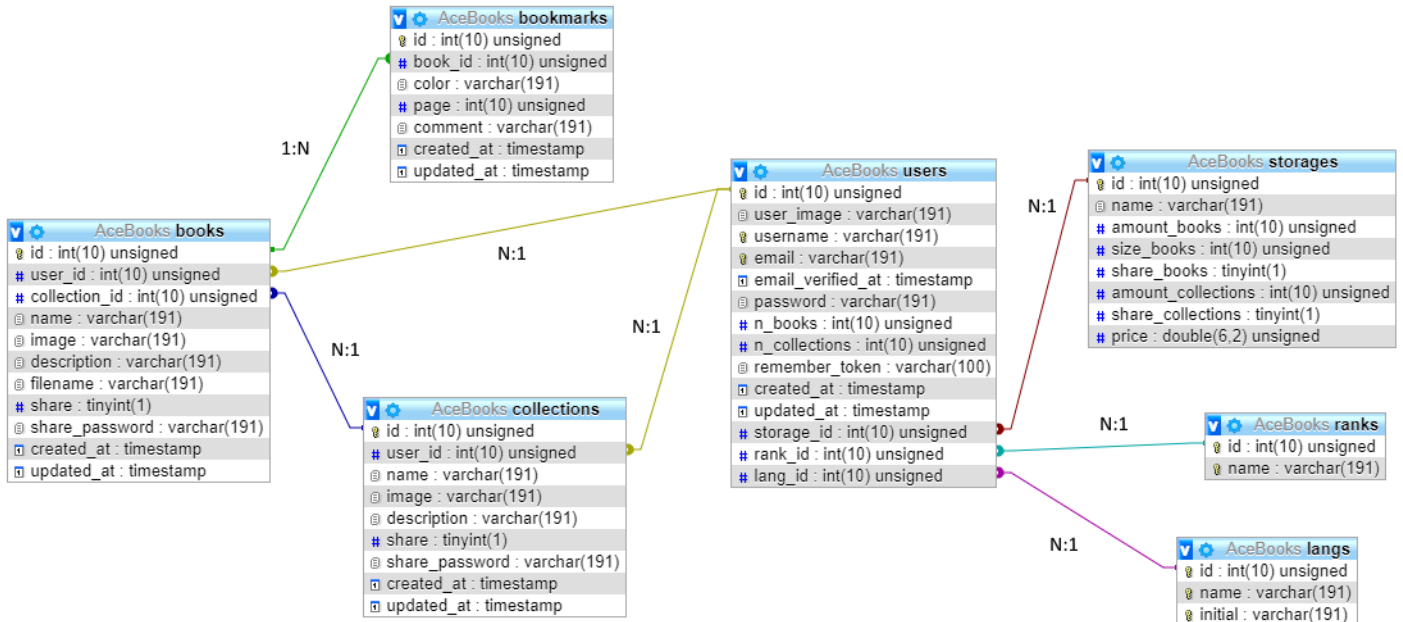
- Procesador: AMD Ryzen 5 2600.
- Placa base: MSI B450-A PRO.
- RAM: 8GB DDR4.
- Tarjeta gráfica: GTX 660 de 2GB.
- Discos duros: 1 TB + 3 TB.

3.2 Software.

- Sistema operativo: Windows 10.
- Editor de código: Visual Studio Code.
- Control de versiones: Git (repositorio en GitHub).
- Para documentar el proyecto: Microsoft Office 2019.
- Editores de imágenes: Gimp y Paint.NET.
- Contenedor de software: Docker (en el he usado PHP 7.3, phpmyadmin, MySQL 5.5 y nginx).
- Gestores de paquetes: npm, yarn y composer.
- Framework en el backend: Laravel
- Framework en el frontend: Vue.JS (usado en momentos puntuales, la mayor parte del frontend es con Blade, el motor de plantillas de laravel).

4. Diseño.

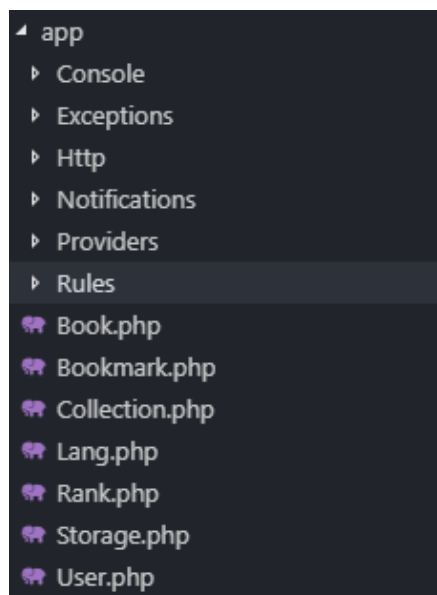
4.1 Diagrama entidad-relación y tablas normalizadas.



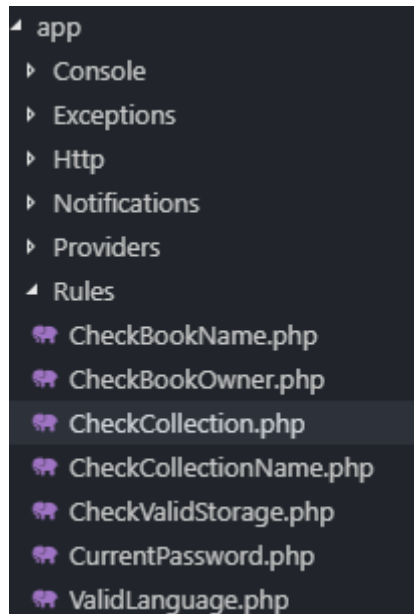
4.2 Estructura de la aplicación.

A continuación, explicaré la estructura del proyecto, me centraré en las partes importantes, ya que laravel tiene su estructura de directorios.

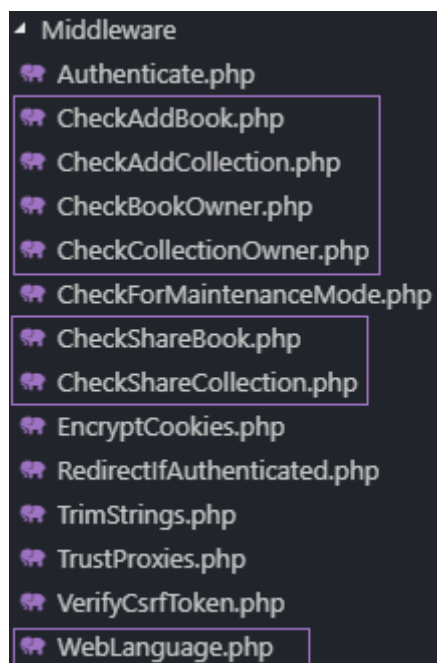
- En **app** tenemos los modelos, estos archivos son los encargados de manipular la información de la base de datos.



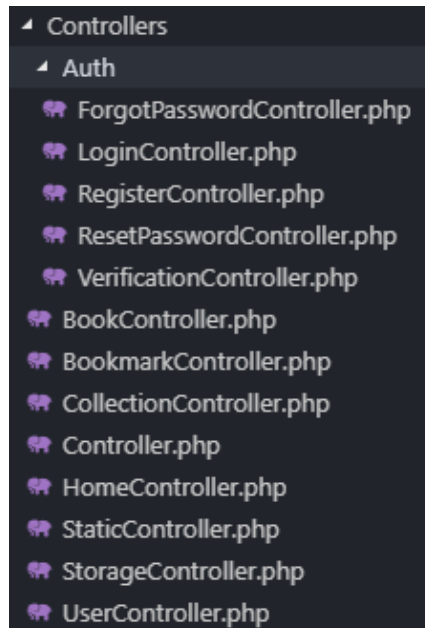
- En **app/Rules** tenemos normas personalizadas para validar los distintos formularios, todos ellos se encargan de comprobar que “x” valor recibido exista en la base de datos y/o el usuario sea el dueño de dichos datos.



- En **app/Http/Middleware** tenemos distintos filtros que se aplicaran al acceder a distintas páginas, por ejemplo, uno que compruebe que el usuario ha iniciado sesión o ha verificado su correo (entre otras cosas), yo he creado los siguientes (resaltados en la imagen), principalmente son para controlar los límites de la tarifa del usuario, además hay otros que se encargan de restringir el acceso a documentos y colecciones que no son de su propiedad.



- En **app/Http/Controllers** tenemos los controladores, estos son los encargados de usar las funciones proporcionadas por los modelos para tratar la información y enviar los resultados a las vistas.

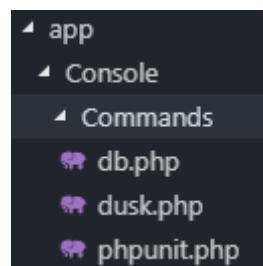


- En **app/Providers** tenemos distintos servicios, yo he creado uno llamado **BladeServiceProvider.php**, crea directivas personalizadas para Blade, el motor de plantillas, los que yo he creado es para mostrar contenidos según el rol del usuario, por ejemplo, los administradores podrán ver un enlace para acceder al panel de control en el “sidebar”.

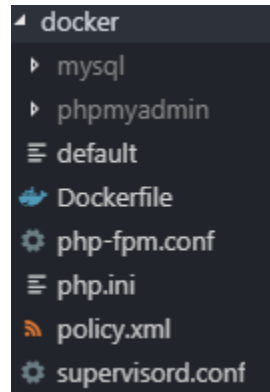
```
public function boot()
{
    Blade::if('rank_more_or_equal', function ($rank_id) {
        return Auth::user()->rank_id >= $rank_id;
    });

    Blade::if('rank_less_or_equal', function ($rank_id) {
        return Auth::user()->rank_id <= $rank_id;
    });
}
```

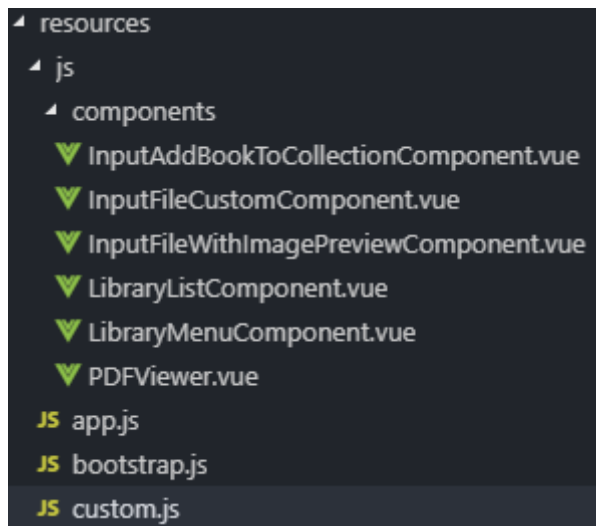
- En **app/Console/Commands** tenemos una serie de comandos personalizados, creados para acelerar las pruebas durante el desarrollo de la página web, el primero se encarga de importar la base de datos, los otros dos son para hacer tests con dusk y phpunit.



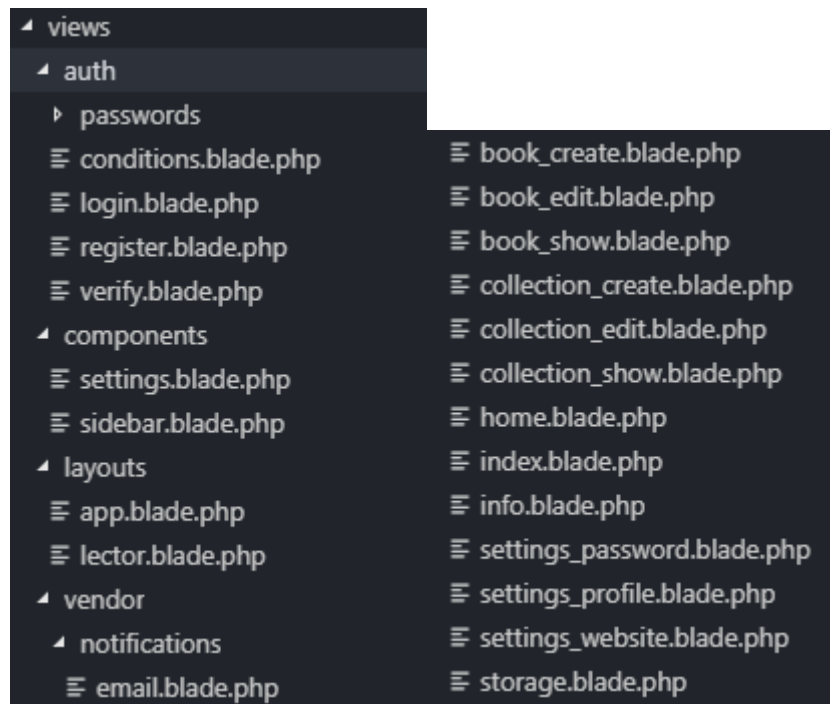
- En **database/migrations** tenemos la estructura de la base de datos y en **database/seeds** una serie de datos base que serán importados junto a la estructura de la base de datos.
- En **docker** tenemos los archivos necesarios para configurar el contenedor usado para desarrollar la página web.



- En **public/images** se almacenarán las imágenes de la web, incluidas las portadas de las colecciones y los documentos.
- En **public/books** se almacenarán los documentos de los usuarios.
- En **resources/js** tenemos los archivos encargados de empaquetar todo el JavaScript de nuestra web en un solo archivo, además de custom.js que será el archivo que contendrá la lógica del sidebar en la versión para pantallas pequeñas.
- En **resources/js/components** tenemos los distintos componentes de Vue.JS

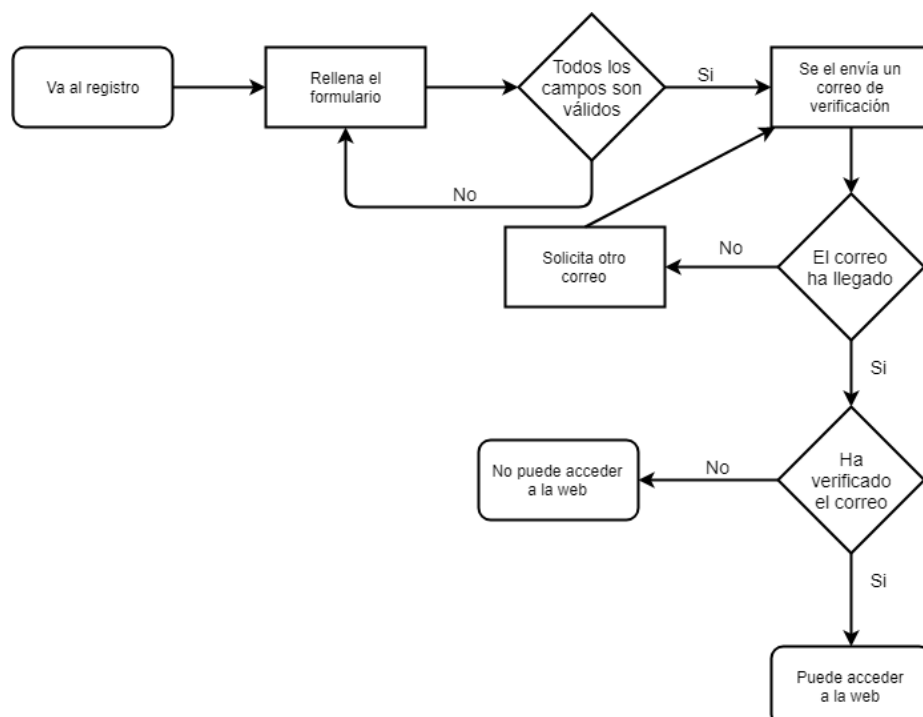


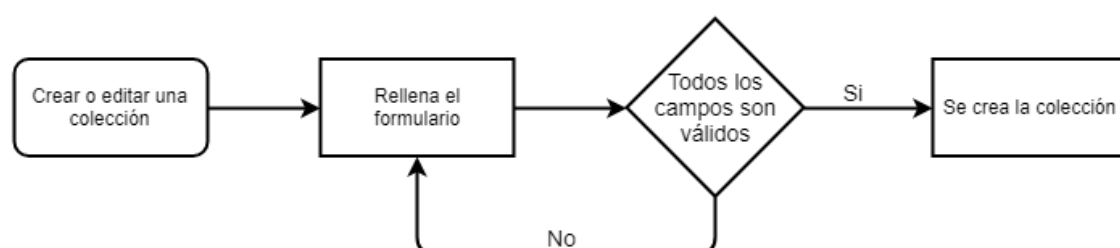
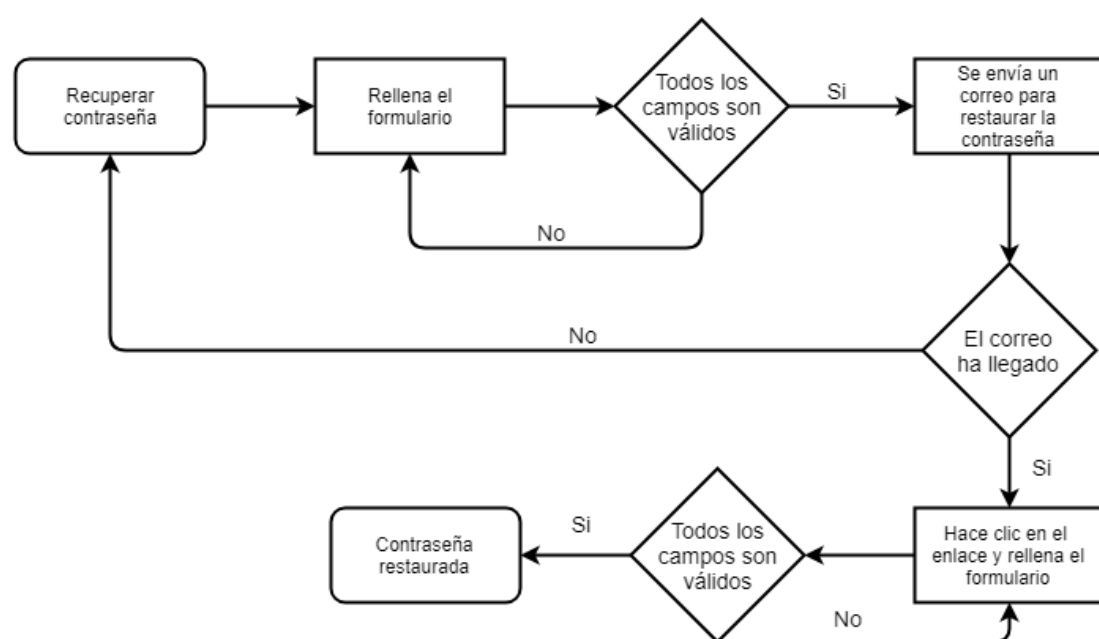
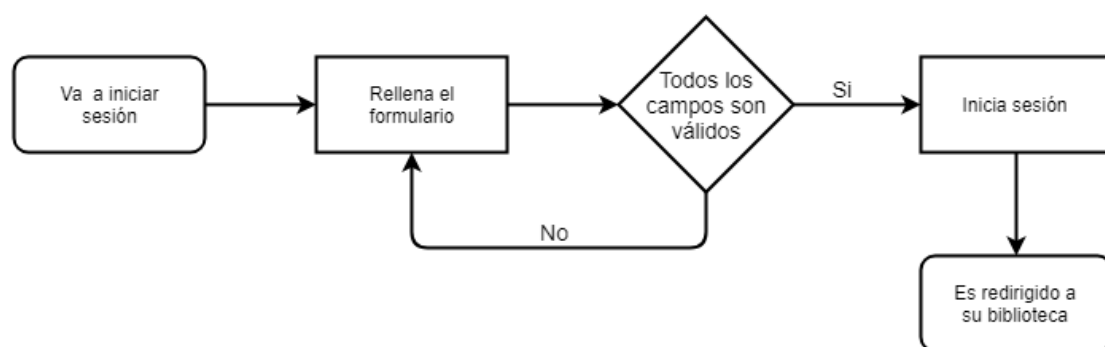
- En **resources/lang** tenemos los distintos archivos que proveen la traducción en castellano e inglés de la página web.
- En **resources/sass** tenemos las hojas de estilos
- En **resources/views** tenemos las distintas vistas de la página web, todas ellas usan el motor de plantillas blade de laravel.

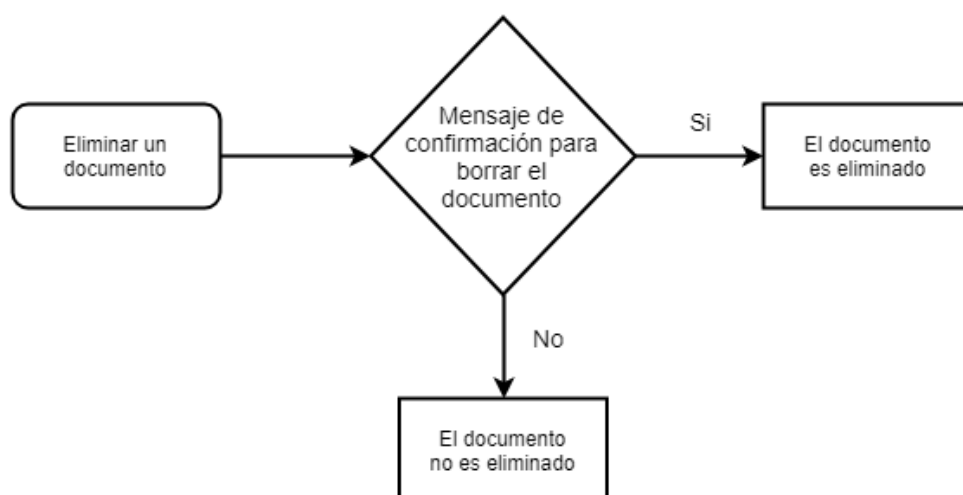
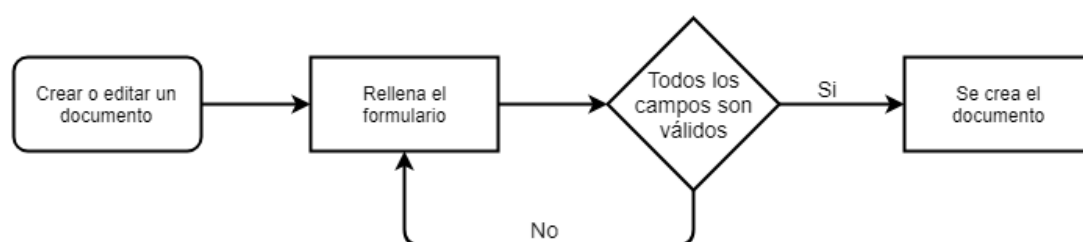
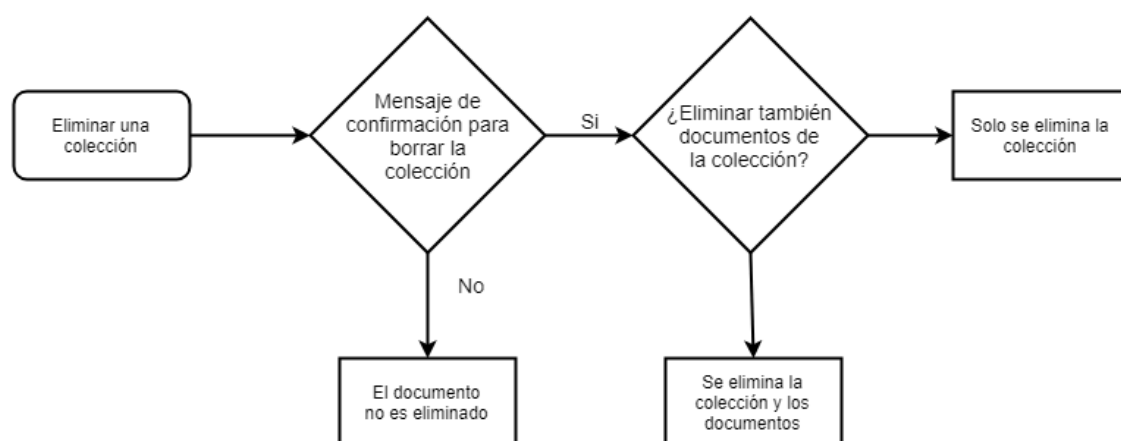


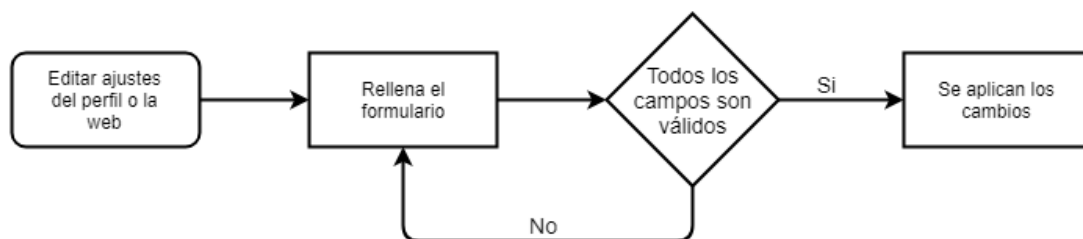
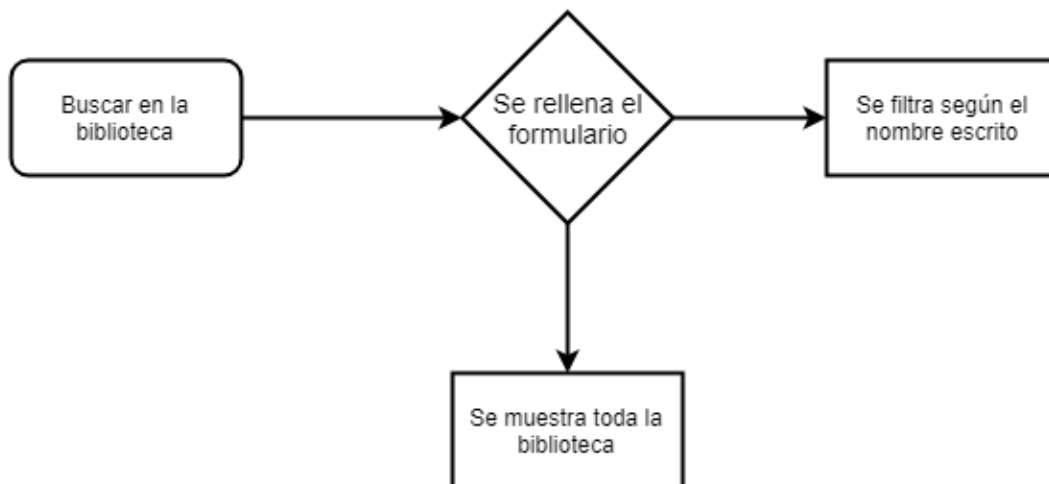
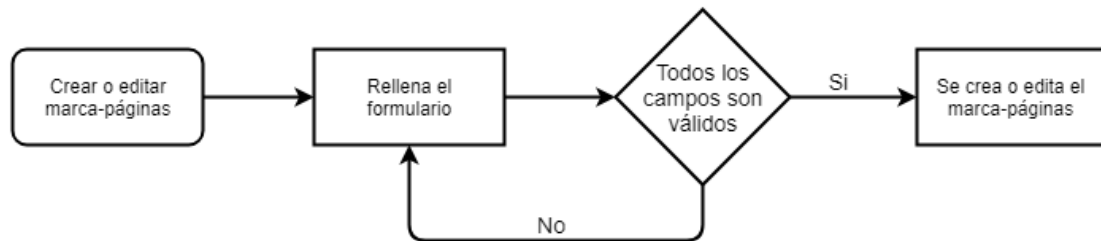
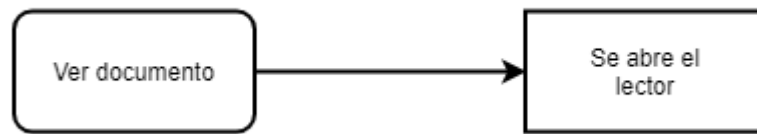
- En **routes/web.php** podemos encontrar todas las rutas de la página web.
- En **la raíz del proyecto** tenemos docker-compose.yml este archivo se encarga de crear el contenedor de docker, usando los ajustes que tiene en su interior y los archivos del directorio docker. Además, tenemos los distintos “.env”, “.env.docker” es usado para iniciar el proyecto con docker, mientras que “.env.example” es para iniciar el proyecto en un servidor o en local usando por ejemplo XAMPP, WAMPP o LAMPP.

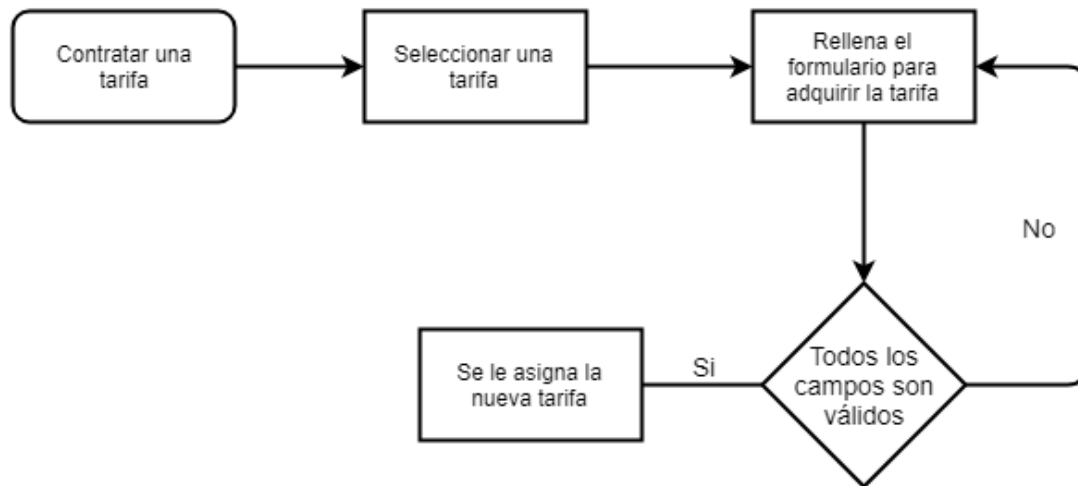
4.3 Flujo de control de cada una de las páginas PHP que componen el proyecto.







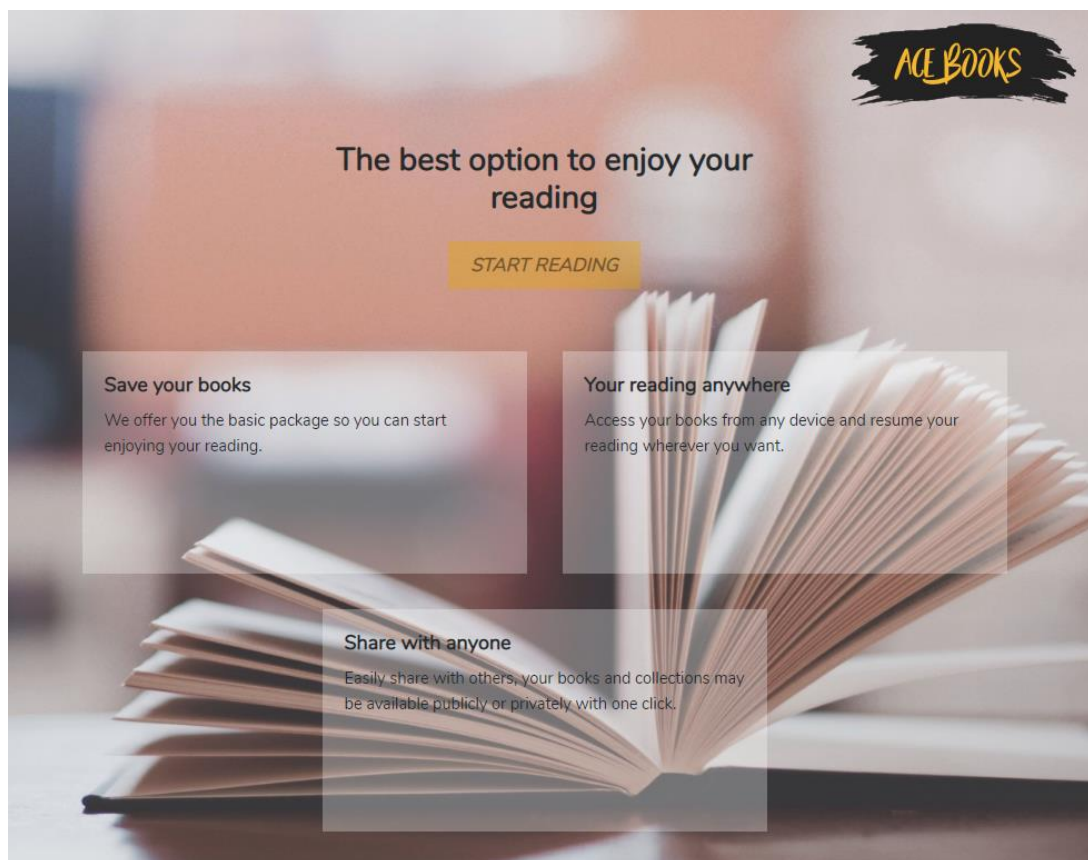




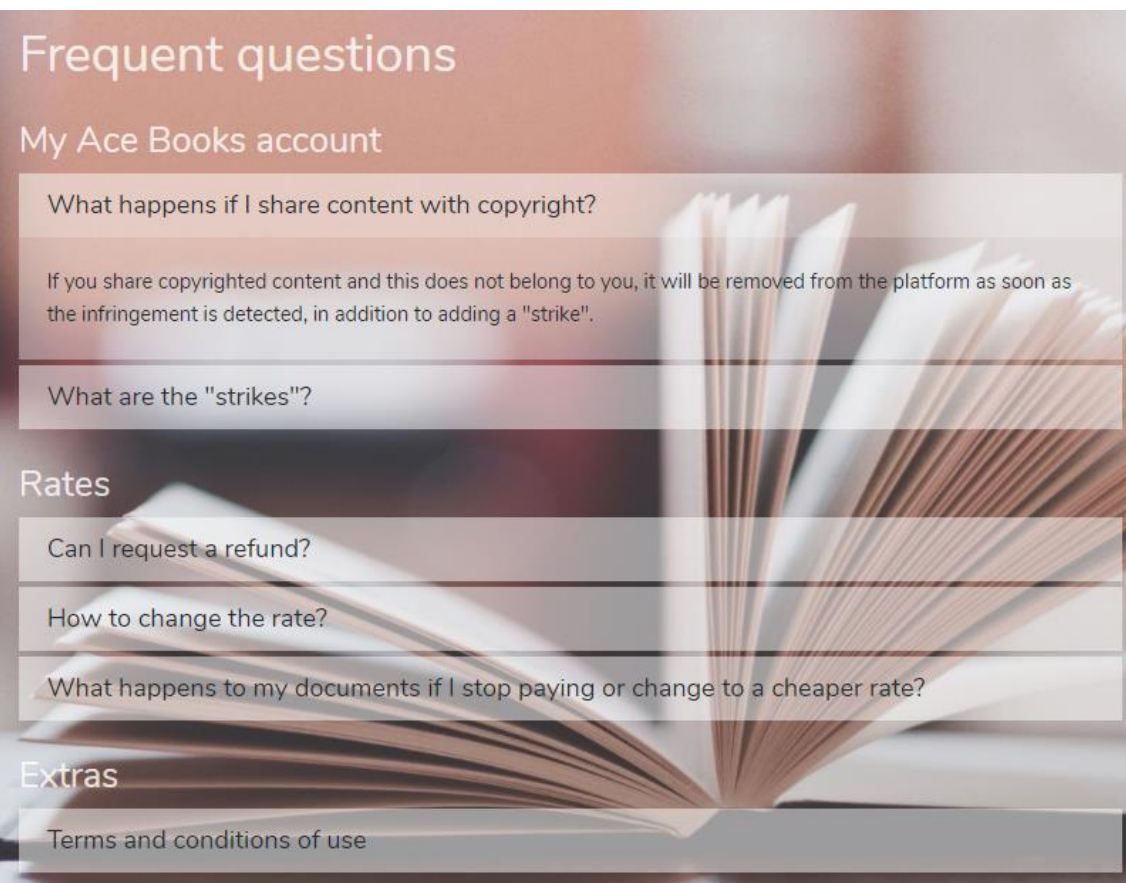
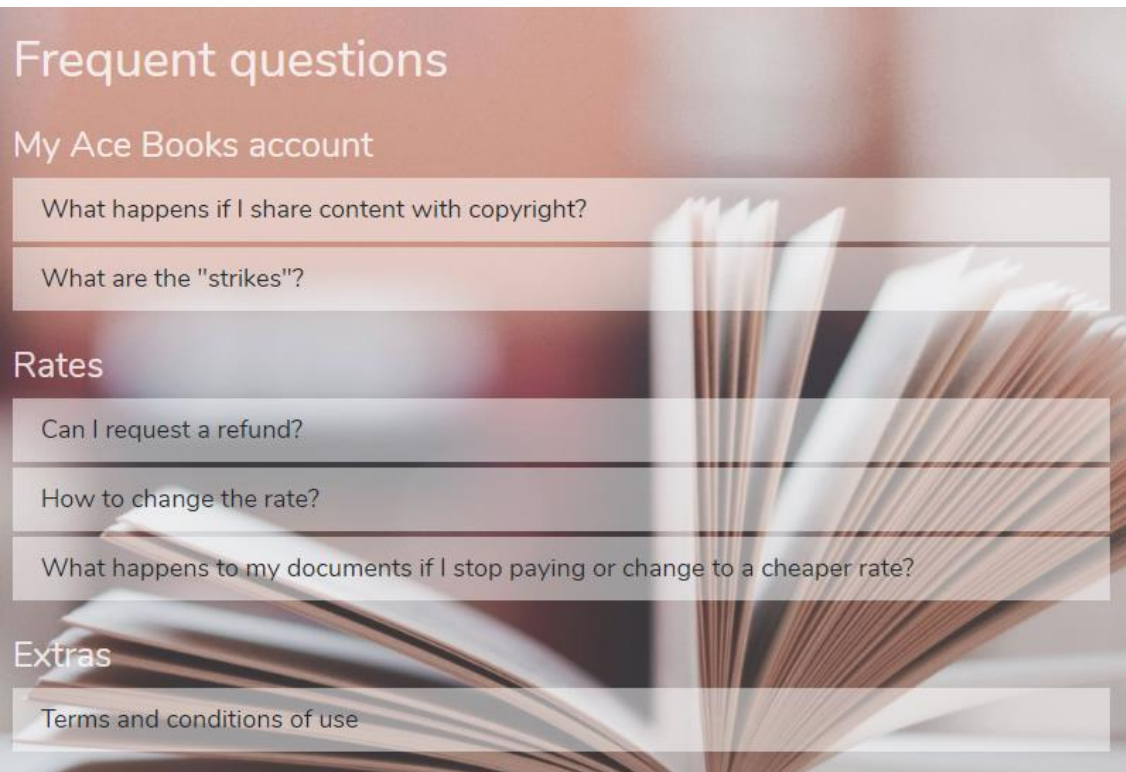
4.4. Interfaz.

A continuación, mostraré el diseño de la web, me he inspirado en la línea de diseño usada por Microsoft, llamada “Fluent Design”, es un diseño que juega con las transparencias y los desenfoques.

- Inicio de la web, se muestra información sobre la web:



- Sección de preguntas frecuentes, he usado el acordeón de bootstrap.




- Sección para adquirir más almacenamiento.

	Package 1 (0 €)	Package 2 (4.95 €)
		<input type="button" value="Buy"/>
Maximum number of documents	10	50
Maximum size of documents	5 MB	10 MB
Share documents	No	No
Maximum number of collections	5	15
Share collections	No	Yes

- Ajustes del perfil del usuario y la web, el diseño está basado en las aplicaciones universales de Windows, separando las opciones del contenido de estas.

Settings

- Edit profile
- Change password
- Web settings



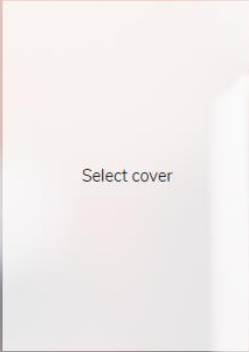
*It is recommended to use an image with equal width and height

- Biblioteca, aquí se muestran las colecciones y los documentos del usuario.



- El formulario de crear y editar tanto en documentos como en colecciones es el mismo, solo cambio los campos requeridos.

Add collection






Select cover

*It is recommended to use the following size, height: 300px | width: 212px

* Name

Description

Book 1 

Select a book  

Add collection

- Por último, quiero enseñar el diseño del lector, todo está en la barra de arriba, ofrece modo cascada y paginado, modos de zoom e incluso marca-páginas.



5. Programación

Respecto a temas de programación no tengo mucho que decir, ya que sabiendo usar laravel podemos interpretar el código muy fácilmente. Aún así, quiero destacar una serie de cosas que he desarrollado. Tanto en el backend como en el frontend.

Empecemos por el backend, lo primero que quiero mencionar es el tema de las traducciones, mi página web está disponible en castellano e inglés, el usuario una vez registrado y con su sesión iniciada puede cambiar el idioma de la web y entonces entra en acción este middleware, lo que hace es comprobar el idioma seleccionado por el usuario y aplicarlo, esto se tiene que hacer cada vez que se accede a una sección de la web.

```
public function handle($request, Closure $next)
{
    $locale = 'en';

    if(Auth::check())
    {
        $locale = Auth::user()->lang->initial;
    }

    App::setLocale($locale);

    return $next($request);
}
```


Tengo muchos middlewares, ya que he querido agregar seguridad y privacidad a mi página web, por ejemplo, este middleware se encarga de comprobar que la colección a la que el usuario intenta acceder le pertenece, todo gracias al método “contains” de las colecciones de laravel, en este caso devolverá “true” si existe una colección con el ID que recibe y “false” si no existe.

```
public function handle($request, Closure $next)
{
    $collection_id = $request->collection->id;

    if($request->user()->collections->contains('id', $collection_id))
    {
        return $next($request);
    }

    return redirect()->route('home')->with('message_limit', trans('collections.ownerError'));
}
```

Ahora vamos a mencionar algunas cosas del frontend, lo primero a mencionar es el tema de mostrar una “preview” de la portada a subir o la imagen del perfil del usuario, lo que hago es ocultar el input encargado de subir la imagen y enviar el foco a la imagen que muestra la “preview”, para actualizar dicha imagen uso el evento onchange de JavaScript, en vue.js es @change=”función”.

```
<input name="image" id="image-upload" class="d-none" type="file" @change="onFileChange" />
```

```
onFileChange(event)
{
    const image = event.target.files[0];
    this.url = URL.createObjectURL(image);
}
```

Otra cosa a mencionar es el uso de axios, una librería para usar ajax de manera más sencilla, esto lo he usado tanto en el buscador de la biblioteca como en los marcadores de las páginas del lector. En el caso del buscador estoy vinculando el input con una variable de vue.js y ejecuta una función cada vez que el usuario termina de teclear, esta función hace una petición con el método POST para obtener los documentos y colecciones del usuario con el título escrito.

```
axios.post('library',
{
    '_token': this.csrf,
    'search': this.search
})
.then(function(response)
{
    let collections = response.data.collections;
    let books = response.data.books;

    collections.forEach(collection => collection.type = "collection");
    books.forEach(book => book.type = "book");

    this.library = collections.concat(books);
})
.bind(this));
```

Y por último el lector, explicaré por encima como funciona todo, lo primero que tengo que decir es que uso un paquete llamado vue-pdf, el enlace a el lo dejo en la bibliografía.

El modo cascada lista todas las hojas del documento, cargando en memoria solo unas 10 páginas, ya que cargar todo el pdf supone un gran consumo de memoria, lo he probado y un pdf de unas 300 páginas puede consumir hasta 2 GB de RAM, el modo paginado solo muestra una página, se da la posibilidad de cambiar de página usando los botones correspondientes o usando el input que indica la página actual.

```
<div
class="mx-auto pdf-page"
v-for="i in numPages"
v-show="mode === 'cascade' || (mode === 'paginate' && i == currentPage)"
:key="i">
  <pdf
    v-if="i >= currentPage - 5 && i <= currentPage + 5 || i === 1"
    :ref="'page' + i"
    :src="src"
    :page="i"
    :style="'width: 100%;'"
    @page-loaded="pageHeight === 0 ? pageHeight = $refs['page1'][0].$el.clientHeight : null"
  />
  <pdf
    v-else
    :ref="'page' + i"
    :style="'height: ' + pageHeight + 'px;'"
  />
</div>
```

El zoom antes era por página, ahora se aplica directamente en el lector, ya que así consume menos recursos. Se usa la propiedad CSS transform: scale();

Para determinar la página actual se calcula el scroll que hay en el contenedor (div) del visor, deajo la línea que lo calcula:

```
// ((Tamaño de la página * escala) * número de páginas) +
// (número de páginas * (margin-button * escala))
let calc = (this.$refs['page1'][0].$el.clientHeight * this.zoom)
* (this.currentPage - 1) + ((this.currentPage - 1) * (10 * this.zoom));
```

Esto se utiliza después de una manera un tanto diferente para calcular el scroll que hay que añadir para pasar página con los botones o el input en el modo cascada.

Y acabamos con los marca-páginas, esto simplemente envía las variables que tenemos en vue.js a laravel usando axios, usando el método POST, gracias a esto creamos un CRUD básico para gestionar el tema de los marca-páginas.

```
addBookmark()
{
  axios.post('/bookmarks/add',
  {
    '_token': this.csrf,
    'book': this.idBook,
    'page': this.currentPage,
    'comment': this.newComment
  })
  .then(function(response)
  {
    this.loadBookmarks();
    this.newCommentStatus = response.data.status;
    this.newComment = '';
  })
  .bind(this);
},
```

6. Conclusiones.

Tras realizar este proyecto, me he parado a pensar que características que debería pulir, incluso cuales añadir (ya hablaré de esto en el siguiente punto).

Cuando tuve que pararme a pensar que proyecto hacer lo tenía más que claro, Ace Books es una idea que siempre ha estado en mi cabeza, he usado un montón de lectores de documentos, pero todos tenían algo que los hacían difíciles de usar, desde consumir muchos recursos hasta faltarles características que para mí eran indispensables.

Tras comenzar el proyecto seguí los plazos que me marcaba, he tenido mis dificultades, el lector ha sido reescrito tres veces, si, como lees, aun así, hay cosas que pulir en él. Incluso más de una vez se me ha pasado por mi cabeza la idea de reescribir parte del backend para que las sesiones funcionen con tokens y así poder crear una aplicación con react native.

He aplicado cosas que me han enseñado en la empresa, una de ellas es docker, me alegro de haberlo implementado, ya que a última hora el servidor del instituto de hizo incompatible con mi proyecto.

Por mi cuenta he aprendido a cómo usar vue.js ya que el lector de documentos requería de este framework, si no hubiese usado esto el código del lector sería muy difícil de mantener.

7. Mejoras.

En lo que mejoras se refiere, me centraría en mejorar el código del lector y pulir ciertos controladores que se han quedado un poco mal organizados, además de terminar el sistema de tarifas. Pero por falta de tiempo esto no ha sido posible.

En cuanto nuevas funciones tengo unas cuantas en mente:

- Reescribir parte del backend y el frontend para que las sesiones funcionen con tokens, usaría laravel passport, esto implicaría que el frontend fuese completamente en vue.js.
- Terminar la sección de tarifas, ofrecer un servicio premium con alojamiento ilimitado para así sustentar los servidores de la página web y ofrecer nuevas funciones a los usuarios.
- Soporte a más formatos, actualmente solo se leen documentos pdf.
- Añadir la posibilidad de compartir documentos y colecciones, esto implicaría añadir la función de reportar contenido que infrinja las leyes de copyright.
- Crear un panel de administración propio, ya que actualmente se usa voyager y este limita la gestión la web.
- Añadir un sistema de notificaciones, se informará al usuario de la retirada de documentos y colecciones compartidas que infrinjan las leyes de copyright, además de proporcionar información de rebajas en las tarifas de la web.
- Posibilidad de recomendar libros a otros usuarios.
- Un sistema para publicar libros, ya sean de pago o gratuitos.
- Añadir tests e integración continua, para evitar subir a producción versiones con errores.
- Eliminar completamente jQuery, se usaría bootstrap junto a vue.js.
- Crear una aplicación de la web con react native, para ello se tendría que realizar antes el primer punto de esta lista.

8. Bibliografía.

Título	Autor/es	Página Web
Axios	Axios	https://github.com/axios/axios
Bootstrap	Twitter	https://getbootstrap.com
Composer	Nils Adermann, Jordi Boggian y otros desarrolladores	https://getcomposer.org/
Docker	Docker, Inc.	https://www.docker.com
Fluent Design	Microsoft	https://www.microsoft.com/design/fluent/
Iconos	Icomoon.io	https://icomoon.io/
JQuery	jQuery Foundation	https://jquery.com/
Laravel	Taylor Otwell y otros contribuidores	https://laravel.com/
Lodash	Lodash	https://lodash.com/
Node.js	Node.js Foundation	https://nodejs.org/es/
NPM	Isaac Z. Schlueter y otros desarrolladores	https://www.npmjs.com/
Pdf.js	Mozilla Foundation	https://mozilla.github.io/pdf.js/
PHP	PHP Group	https://www.php.net
Popper.js	FezVrasta y otros contribuidores	https://popper.js.org/
Sass	Hampton Catlin, Natalie Weizenbaum, Chris Eppstein, Jina Anne, entre otros contribuidores.	https://sass-lang.com/
Voyager	The control group	https://laravelvoyager.com/
Vue.JS	Evan You	https://vuejs.org/
Vue-pdf	FranckFreiburger	https://github.com/FranckFreiburger/vue-pdf
Yarn	Yarn	https://yarnpkg.com/es-ES/