

Figures Collector

Curso: 2º Desarrollo Aplicaciones Web

2. Índice

1. Portada	1
2. Índice	3
3. Resumen	5
4. Introducción	7
5. Objetivos y características del proyecto	9
6. Finalidad	11
7. Medios materiales usados: humanos, hardware, software	13
8. Planificación del proyecto	15
8.1 Scraper	15
8.2 Base de datos	21
8.3 Clases Utility y Request	27
8.4 Aplicación web	28

8.4.1 Elementos comunes	29
8.4.2 Página search	33
8.4.3 Página index	38
8.4.4 Página producto	39
8.4.5 Páginas de login y alta de usuario	43
8.4.6 Página perfil	44
9. Fase de pruebas	47
10. Conclusiones y trabajos futuros o posibles mejoras	49
12. Referencias bibliográficas	51

3. Resumen

Figures Collector es una aplicación online que permite la gestión y control de una colección de figuras, permitiendo añadir y quitar figuras al perfil del usuario, recibir sugerencias personalizadas y buscar figuras siguiendo distintos criterios de búsqueda y con un poderoso motor de búsqueda basado en tags.

4. Introducción

Las colecciones de figuras de super héroes, anime y videojuegos son uno de los negocios más relevantes en el mercado actual. Hoy en día todos los aficionados de las franquicias más conocidas del planeta quieren figuras lo más detalladas posibles que representen con fidelidad a sus personajes favoritos.

Normalmente el coleccionista busca figuras relacionadas con una licencia o una colección concreta, pero al existir diversos fabricantes, es muy difícil encontrar un catálogo homogéneo.

Además, aunque existan páginas que gestionen colecciones, sus motores de búsqueda no suelen estar a la altura y es muy difícil encontrar un listado fiable de todas las figuras de una determinada franquicia

5. Objetivos y características del proyecto.

- El objetivo de la aplicación es crear un catálogo en crecimiento de todos los productos ofertados por las diferentes compañías y que permita al usuario marcar los productos que ya tiene, así como licencias o colecciones de su interés.
- Poder gestionar desde cualquier dispositivo una colección de figuras de manera rápida y sencilla.
- Crear una base de datos actualizada de las figuras que hay en el mercado. Los datos se mostrarán a través de una web y las figuras se podrán filtrar por distintas categorías.
- Crear un motor de búsqueda lo suficientemente potente como para

6. Finalidad

Crear una aplicación de gestión de colecciones sobre la que poder crear una comunidad online de coleccionistas, así como ampliar el espectro de la aplicación y convertirla en un motor de búsqueda para tiendas que permita al usuario encontrar figuras al mejor precio.

7.-Medios materiales usados:

Scraper: Python 3.7

Aplicación web:

- PHP 7
- MySQL
- MySQL Workbench
- HTML5
- CSS3

Común: Visual Studio Code 1.56

8.-Planificación del proyecto:

8.1 Scraper

Con la primera parte del proyecto, el objetivo era no sólo mostrar un proyecto con una base de datos simulada y con unos pocos registros: Se pretendía crear una tabla completa, con casi 7.000 registros, generada a través del scraping de un site. En este caso, se realizará el scraping de la web goodsmile, almacenando todos sus productos, desde 2005 hasta hoy.

Para ello, se creó una clase scraper en Python, lenguaje que cuenta con dos librerías muy potentes y que facilitan enormemente la tarea: La librería Requests y la librería BeautifulSoup.

A continuación, se explicará la función de cada una de ellas:

La librería Requests permite realizar peticiones HTTP de forma sencilla a través de un script, por lo que sin necesidad de un navegador es posible obtener la respuesta del servidor. Aparte de la URL sobre la que pretendamos hacer la petición, también es posible configurar otros aspectos de la misma, como por ejemplo:

- El tipo de petición que se va a realizar, generalmente GET o POST.
- El encabezado de la petición: Tipo de formato que se envía, tipo de formato que se espera de vuelta, codificación...
- Los parámetros enviados en la URL así como el cuerpo de la petición

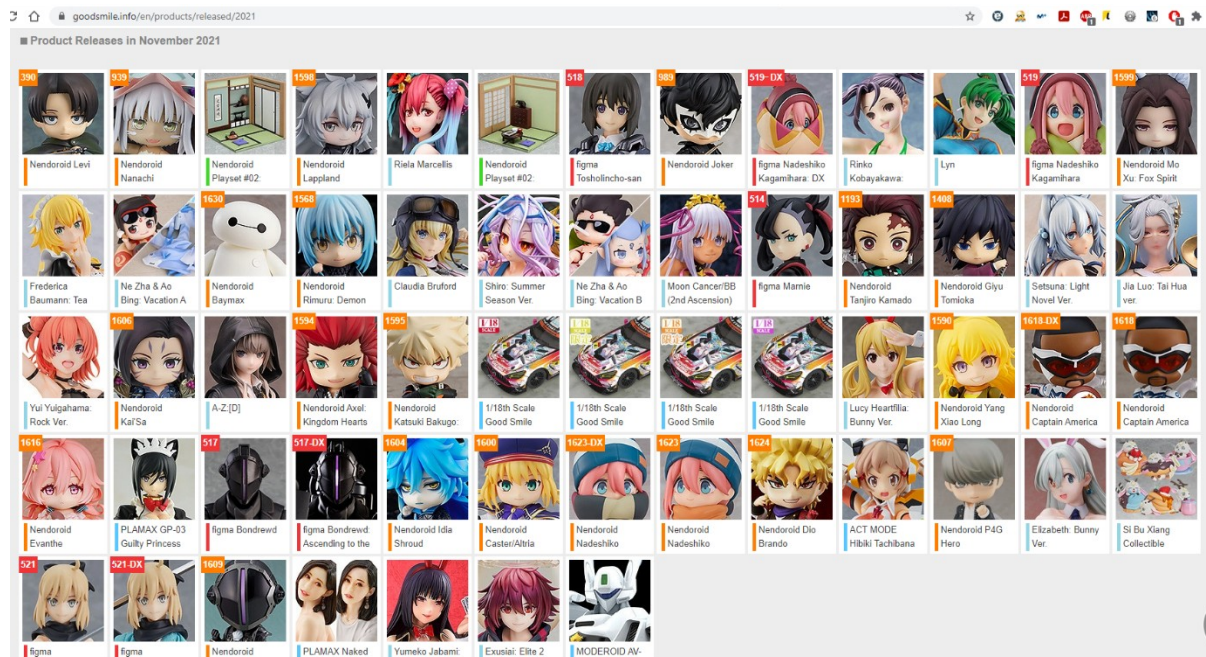
La librería BeautifulSoup permite interpretar el código HTML almacenado en una variable de tipo de String, convirtiéndolo en un formato especial (Tag) a través del cual podemos navegar por su jerarquía, buscar etiquetas con un determinado atributo, obtener el texto y/o los atributos que contiene cada etiqueta, etc...

De esta forma, mediante la librería Requests recuperaremos toda la información que devuelve la página y con BeautifulSoup disgregaremos y extraeremos solo la información que consideremos necesaria.

El funcionamiento del scraper será el siguiente. Es importante señalar que este proceso, aunque siga unas pautas, no es universal para todos los sites del mundo: No solo cada site está estructurado de una manera, además de la creciente aparición de sistemas anti-scraping, que bloquean aquellas peticiones que consideran sospechosas para su servidor y requieren de herramientas extra para obtener esa información, como pueden ser los proxies.

```
year = 2005
main_url = 'https://www.goodsmile.info/en/products/released/'
cookie = {'age_verification_ok': 'true'}
```

En primer lugar, se realiza la configuración básica de la web a scrapear. Como solo queremos obtener la información de una página HTML que no realiza peticiones extra a un servidor, será suficiente con hacer peticiones GET a la dirección que agrupa los productos por año. Para ello solo será necesario configurar en que año queremos empezar las peticiones (En nuestro caso, 2005), configurar la dirección principal y añadir una cookie que inutilizará los sistemas de control parental.



Ejemplo de página de resultados a scrapear (Fuente: www.goodsmile.com)

```

outfile = open('goodsmile.json')

with open('goodsmile.json', 'r') as f:
    outfile = f.read()

if outfile:
    product_list = json.loads(outfile)
else:
    product_list = []

```

A continuación, abrimos un archivo de tipo JSON, que será donde se almacene toda la información que extraigamos. Si el archivo tiene datos, recuperará toda la información añadida en él anteriormente. En caso contrario, creará una lista vacía que cumplirá la misma función.

```

id_list = []

for p in product_list:
    id_list.append(p.get('id'))

```

Este fragmente de código es importante, ya que crearemos una lista con todos los ids de los productos almacenados en el documento JSON. En caso de que el documento estuviese vacío, no hará nada. Más adelante veremos la función de esta lista.

```
while year < 2023:
    print('*'+ str(year)+'*')
    r_url = '%s%s' % (main_url, year)
    r = requests.get(r_url)
    soup = BeautifulSoup(r.text, 'lxml')

    products = soup.find_all('div', {'class': 'hitItem'})
```

A partir de aquí dará comienzo el bucle, y es que realizaremos una petición HTTP por cada año hasta 2023 (Año para el cual ya no aparecen productos). Requests realizará la petición y del resultado, BeautifulSoup la convertirá en lo que comúnmente es conocido como “sopa”. A partir de ahí, podremos buscar los elementos que deseemos. En este caso, queremos todos los elementos div con clase “hitItem”, ya que cada uno de ellos se corresponde con un producto de la página.

```
for p in products:

    url = p.find('a').get('href')
    id = re.findall(r'\d+', url)[0]

    if id in id_list:
        continue
```

Lo siguiente será realizar un bucle for para extraer información de cada producto. No toda la información que necesitamos está en la web de resultados, así que será necesaria buscar la url para empezar a realizar peticiones por cada producto. De esta URL podremos extraer también el id del producto mediante expresiones regulares. Ahora es cuando comprobaremos si el producto con el que estamos trabajando ya ha sido añadido, buscándolo en la lista de ids creada anteriormente.

```

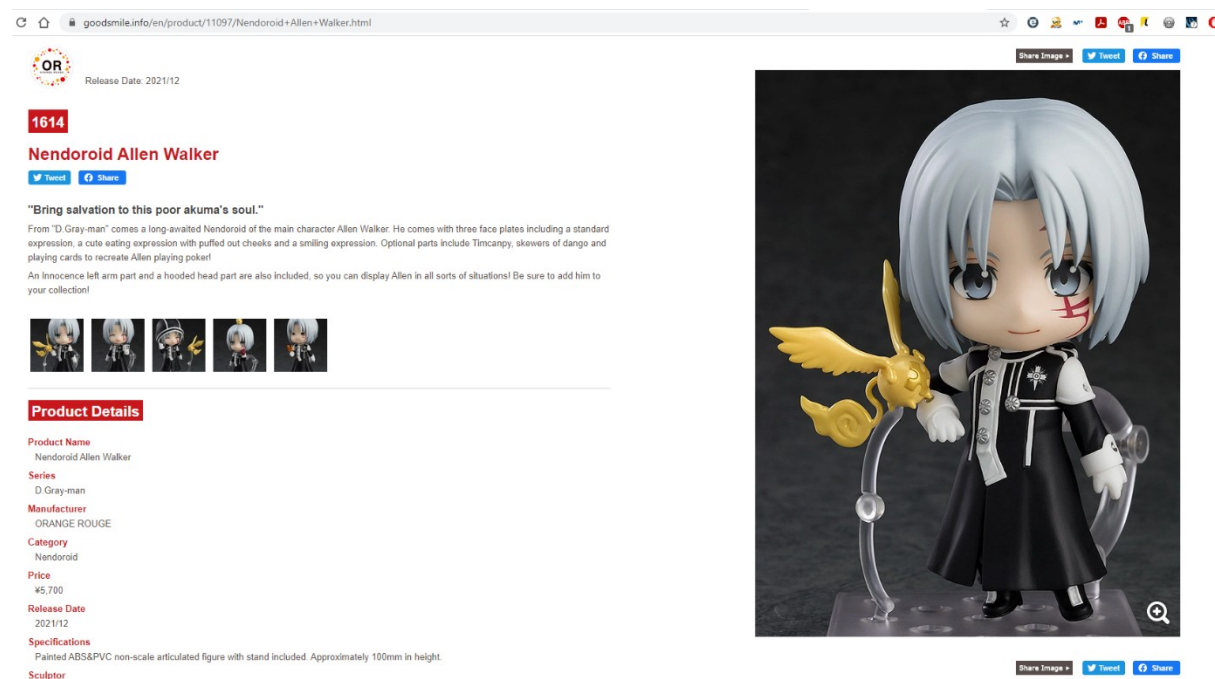
r = requests.get(url, cookies=cookie)
soup = BeautifulSoup(r.text, 'lxml')

name = soup.find('title').text

product_data = soup.find(id='itemBox')

```

Con la URL, ya sí podremos realizar una petición a la página específica del producto. Una vez tengamos la sopa del producto, se buscará un id común a todas las páginas de productos, conocido como itemBox, que contendrá toda la información necesaria posible del producto.



Ejemplo de página de producto a scrapear (Fuente: www.goodsmile.com)

```

if not product_data:

    goods = soup.find('div', {'class': 'goodsblock'})

    if goods:
        goods_data = goods_scraper(goods)
        goods_product = product_base | goods_data
        product_list.append(goods_product)
        continue

```

¿Pero que pasa si no se encuentra dicho id? Hay algunas páginas que tienen un diseño especial, ya sea porque almacenan un tipo de producto que no es una figura o por querer destacar algunos productos concretos. En esos casos, será necesario buscar otras etiquetas y scrapear la página de manera diferente. Fue necesario realizar 12 scrapers adicionales para cubrir cada tipo de página especial.

```
product = {}

product['url'] = url
product['id'] = id
product['thumbnail'] = thumbnail

.....

product_list.append(product)
time.sleep(3)

year += 1
with open('goodsmile.json', 'w') as outfile:
    json.dump(product_list, outfile)

outfile.close()
```

El resto del proceso se culmina buscando en la sopa toda la información restante que queremos del producto: Descripción, año, imágenes, serie, tipo de figura, precio... Una vez lo tenemos todo, el producto se añade a una lista, se realiza una pausa de 3 segundos para evitar baneos por excesivas peticiones. Una vez añadidos todos los productos, se guardan en el archivo JSON y se comienza el scrapeo del siguiente año. El proceso se repetirá hasta llegar a la fecha marcada como límite.

Nótese que este será el único script realizado en Python, ya que para la creación de la aplicación solo utilizaremos PHP como lenguaje de servidor.

8.2 Base de datos

Una vez hemos almacenado toda la información de los productos en un documento de texto, lo siguiente será almacenarlos en una base de datos para poder trabajar con ellos. Aunque existan modelos de bases de datos no relacionales que ya trabajan directamente con JSON, apostamos por trasladar toda esta información a una base de datos de MySQL, ya que todos los productos contienen los mismos campos (aunque en algunos casos estuviesen con valor nulo).

El modelado de la base de datos sería el siguiente:

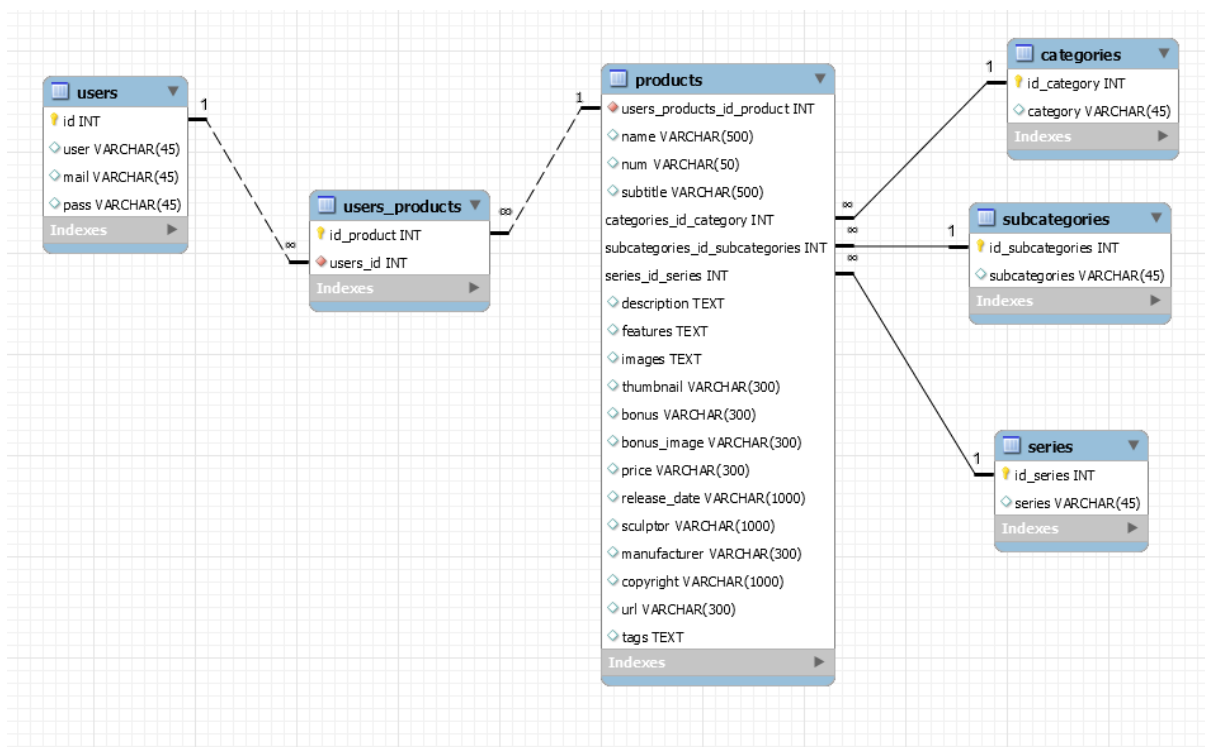


Diagrama base de datos (Fuente propia)

Existiría una tabla principal, que sería products, cuyos campos serían todos aquellos creados en el JSON generado en el scraper. De dichos campos, se podrían sacar 3 tablas para normalizarlos, que serían: Categorías, Subcategorías y Series. Por otro lado, existiría una tabla usuarios que almacenaría a todos los usuarios registrados en la aplicación. Por último,

una última tabla relacionaría los usuarios con los productos que tienen, siendo ambas ids claves primarias. Todas estas relaciones serán de 1:N.

Ahora vamos a repasar las clases que gestionan la base de datos.

```
class DB{

    private $connect;
    private $query;
    private $total = 0;

    function __construct($host, $user, $pass, $schema, $port){
        $this->connect = new mysqli($host, $user, $pass, $schema, $port);
        $this->error();
    }

    function setQuery($query){
        $this->query = $this->connect->query($query);
    }
}
```

Empezamos por la clase DB, una clase “madre” desarrollada para gestionar las conexiones y las queries de cualquier clase que la herede. El objeto se construye con la creación de una conexión mysqli usando los valores que se le pasan por parámetros.

```
if ($this->query){
    if (is_object($this->query)){
        $this->setTotal();

        return $array;
    } else {
        return $this->query;
    }
}
```

Su función básica y más importante es runQuery, la cual ejecuta la query, setea el resultado como atributo de la clase a través de setQuery. Una vez hecho esto, se verifica el tipo del resultado: Si es un objeto, significa que la query realizada es un SELECT y que está

devolviendo un objeto que contiene los resultados de la búsqueda. En caso de no ser un objeto, significa que no tiene resultados que mostrar, ya sea porque la consulta no tiene por qué devolver resultados (Como ocurre con los INSERT, UPDATE y DELETE), porque el SELECT ha devuelto una tabla vacía, o porque ha habido un error en la ejecución de la query.

```
$array = [];  
  
while ($result = $this->query->fetch_assoc()){  
    if ($field){  
        $array[] = $result[$field];  
    } else {  
        $array[] = $result;  
    }  
  
}  
  
if ($unique && $array){  
    $array = $array[0];  
}
```

En el caso de que la query sea un SELECT que devuelve resultados, vamos a recorrer todas las filas y vamos a añadirlas a un Array. La función runQuery cuenta con dos parámetros opcionales para personalizar directamente en la función el array que se va a devolver. Si no usamos ninguno, se devolverá un array de array asociativos. En caso de que usemos field, solo devolveremos en un array la columna indicada en dicho parámetro. Si utilizamos el parámetro unique, devolveremos solamente el primer array asociativo del listado. Esto lo utilizaremos cuando tengamos la certeza de que vamos a sacar un único resultado (Por ejemplo, un SELECT que busque por ID), y por tanto queramos trabajar directamente con el Array asociativo resultante.

```

class Con extends DB{

    private $host = 'localhost';
    private $user = 'root';
    private $pass = '';
    private $schema = 'figures_collector';
    private $port = 3306;

    function __construct(){
        parent::__construct($this->host, $this->user, $this->pass, $this->schema, $this->port);
    }
}

```

Para evitar tener que pasar los parámetros de conexión a cada clase que herede de DB, vamos a crear una clase intermedia llamada Con (Conexión), que tendrá como atributos los parámetros de conexión a la base de datos. Esta clase Con será heredada por otras dos clases en nuestra aplicación: La clase FC y la clase Users.

La clase FC (Abreviatura de Figures Collector) será la encargada de gestionar toda la operativa de base de datos relacionada con todos los productos guardados en la base de datos. Ese será el punto de partida de dicha clase, ya que la primera clase que se creará para esta clase será la que inserte todos los productos de la base de datos.

```

include('figures_collector.php');

$fc = New FC();

$json = file_get_contents("products.json");

$array = json_decode($json, true);

foreach ($array as $a){
    $fc->insertProduct($a);
}

```

Esto se realizará con un sencillo script en PHP podremos abrir el JSON, convertir el contenido en un array asociativo y añadir cada producto mediante un bucle en la tabla

Products. La misma función de insertProduct será la que traducirá todo el array asociativo en distintas variables que serán incluidas en la query del INSERT.

```
foreach ($tags as $key=>$value){
    if ($key == count($tags)-1){
        $condition .= "tags LIKE '%$value#%'";
    } else {
        $condition .= "tags LIKE '%$value%' AND ";
    }
}
return parent::runQuery("SELECT id, name, thumbnail FROM products
WHERE $condition ORDER BY id_category, subcategory, id DESC LIMIT $offset, $resultsPerPage");
```

Otras funciones de la clase FC será la búsqueda de productos según distintos campos: Id, Serie, Categoría, Subcategoría, Tags... En muchas de estas funciones, pasamos un array como parámetro, por lo que será necesario modificar la query de forma dinámica para que aplique tantas cláusulas WHERE como elementos vengan en el Array.

Hemos mencionado la búsqueda por tags, una columna en la que cada producto tendrá asociado unas palabras clave para realizar búsquedas. Sin embargo, estas palabras claves no se han extraído directamente desde el scraper, por lo que la columna está completamente vacía. Para rellenar esta columna en cada producto, procederemos a la creación de un generador de tags.

```
foreach ($product as $key => $value){
    $value = str_replace($bad_chars, ' ', $value);
    $value = str_replace('-', '', $value);
    $tags_candidates = explode(' ', $value);
}
```

El funcionamiento del generador es el siguiente: En primer lugar, se hará un SELECT de todos los productos que mostrará los campos que contengan aquellas palabras clave que queremos guardar como tags: Nombre, Series, Categoría, Subcategoría, Fecha de lanzamiento

y Copyright. El siguiente paso será eliminar todos aquellos caracteres de las cadenas que no queremos guardar como tags: Signos de puntuación, guiones, símbolos, etc... Una vez tengamos la cadena limpia, la convertiremos en un array de strings mediante explode.

```
if ($key == 'copyright'){
    foreach ($tags_candidates as $c){
        if (is_numeric($c) || in_array($c, $bad_tags) || !
mb_check_encoding($c, 'ASCII')){
            continue;
        }
    }
}
```

Para simplificar, de la fecha de lanzamiento solo cogeremos el año, mientras que del copyright vamos a intentar evitar todas aquellas palabras que no tienen valor como palabra clave. Por ejemplo, si el copyright rezase “Licensed by Nintendo”, de ahí solo nos interesa “Nintendo”. Aunque la página funcione con codificación UTF-8, tampoco incluiremos caracteres japoneses.

```
foreach ($tags_list as $tag){
    $tag = mb_strtolower($tag);
    if (!in_array($tag, $tags) and (!empty($tag))){
        $tags[] = $tag;
    }
}

$id = $product['id'];

$tagStr = implode('###', $tags);
$tagStr = '###'.$tagStr.'###';
$fc->updateTags($id, $tagStr);
```

Finalmente, se verificarán todos los tags obtenidos para cada producto, se guardarán en minúscula y se empezarán a almacenar en nuevo listado, que excluirá todos aquellos tags repetidos o vacíos. Para terminar, el listado de tags se convertirán en una única cadena, en la que cada tag estará separado de 3 almohadillas (#), para facilitar posteriormente su separación

en las búsquedas. Esta cadena se actualizará en la columna tags de cada producto mediante su id.

La otra clase que gestione la base de datos será la de Users. Esta clase no solo controlará las acciones típicas de un usuario, como son el alta y los inicios y cierres de sesión, también anotará todos los productos que hayan sido elegidos por el usuario.

8.3 Clases Utility y Request

Antes de entrar de lleno en la aplicación en sí, vamos a explicar el funcionamiento de dos clases auxiliares que realizan funciones específicas durante el programa, las cuales evitarán una repetición excesiva de código en el programa.

La clase Utility es una clase estática compuesta por un conjunto de utilidades que sirven de apoyo a la creación de elementos dinámicos. Para esta aplicación solo utilizaremos la función estática arrayToSelect.

```
if (isset($selected) && $selected == $element){
    $html .= '<option value="'. $element .'" selected>' . $element . '</option>';
} else {
    $html .= '<option value="'. $element .'">' . $element . '</option>';
}
```

Como su propio nombre indica, esta función permite la creación de etiquetas Select a partir de un Array mediante el uso de un bucle for each. En cada pasada del bucle se crea una etiqueta option, cuyo valor y contenido vendrá determinado por los parámetros de entrada que se le pasen a la función, como por ejemplo marcar con el atributo selected.

```

function __construct($response){
    $this->response = $response;
}

function validate($field){

    $this->validated = true;

    if (!isset($this->response[$field]) || empty($this->response[$field])){
        $this->validated = false;
    }
    return $this->validated;
}

function get($field){
    if ($this->validate($field)){
        return $this->response[$field];
    } else {
        return null;
    }
}

```

La clase Request (No confundir con la librería Request utilizada previamente en Python) se utilizará principalmente para llevar el control de todas las peticiones recibidas por HTTP. Por normal general, se inicializará con una de las super globals \$_GET o \$_POST, y mediante el método get, devolverá una vez haya comprobado que están inicializadas y no están vacías. En caso de no ser así, devolverá null.

8.4 Aplicación web (PHP + HTML + CSS)

En este punto se va a tratar todo lo relacionado con la estructura y estilo código de servidor de la aplicación . Pese a ser lenguajes completamente distintos (Lenguaje de servidor, lenguaje de marcado y lenguaje de estilo respectivamente), la enorme interrelación que tienen PHP, HTML y CSS nos invitan a analizarlos conjuntamente en cada elemento de la web.

Para una mayor organización, comentaremos por separado los principales elementos de cada página de la web.

8.4.1 Elementos comunes

Existen una serie de elementos generales que nos van a acompañar a lo largo de la web. Su existencia ayuda a crear una mayor cohesión, entrelaza todo el sitio y hace posible que podamos acceder a cualquier página desde cualquier página.

Todas las páginas del site están compuestas por el mismo contenedor central, en el que se alojarán los siguientes elementos: Mensaje de bienvenida, logo, barra de búsqueda, menú, y contenido de la página.



Logo (Fuente propia)

El logo del site cumple dos funciones: Además de ser la imagen de marca del site, también sirve como enlace para llegar a la página principal (Index). Al ser bastante accesible, no será necesario hacer un botón específico de Home para el menú.

```
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Poppins">  
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lobster">
```

El logo utiliza la fuente de Google Lobster, siendo el único elemento de toda la web que la utiliza. El resto de la web usa la fuente Poppins, también de Google. Ambas fueron importadas directamente de la api de Google mediante etiquetas links en el Head.

Welcome! Please [login](#) or [sign up](#)!

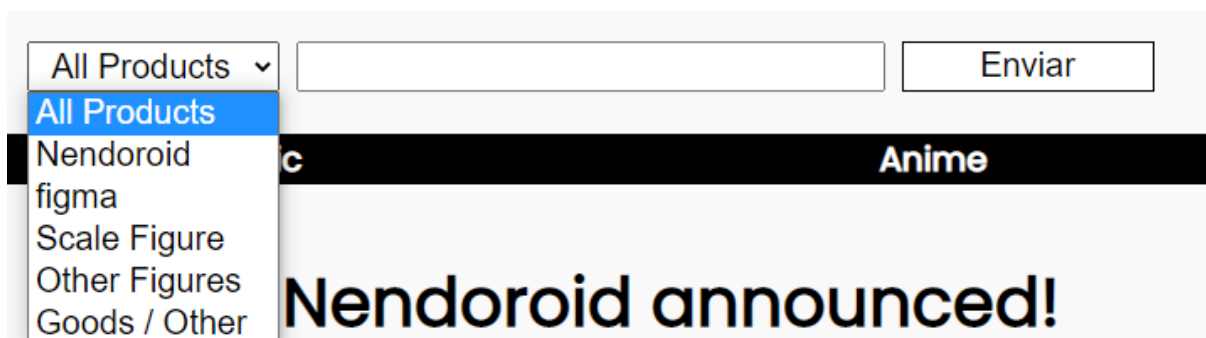
Welcome koto! See your [profile](#) or [close](#) your session

Mensajes de bienvenida (Fuente propia)

Justo encima del logo, la web realiza un mensaje de bienvenida. Este mensaje es dinámico y dependerá de si el usuario ha iniciado sesión o no. En caso de que no exista una sesión iniciada, invitará al usuario a iniciar sesión o a crearse una cuenta. En cambio, si el usuario ya tiene una sesión iniciada, le permitirá ver su perfil o cerrar la sesión. Pese a que pueda pasar desapercibido, este mensaje de bienvenido hace las funciones de pequeño menú e interconecta todos los elementos del usuario de la web.

```
<form action='./session.php' name='close_session' id='close_session' method='post'><input type='hidden' name='type' value='close'>
    <a href='javascript:{}' onclick='document.close_session.submit();'>close</a></form> your session</span>
```

Cabe destacar que el enlace “Close session” no es un enlace al uso, ya que está dentro de un formulario y realiza las funciones de un button o un input submit. Este enlace ejecuta una sencilla función de JavaScript al clickarse, la cual inicia la petición HTTP según el nombre del formulario. Este formulario se dirige a un script auxiliar, session.php, del que hablaremos más adelante.

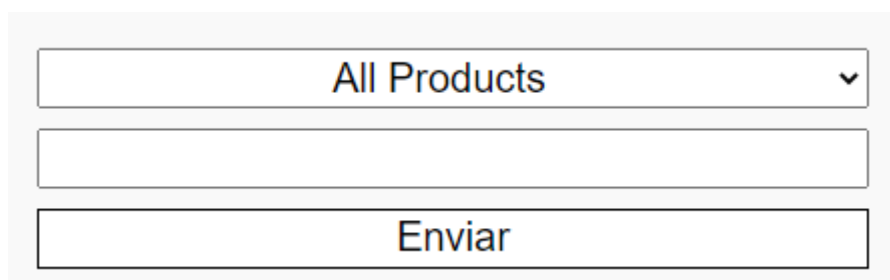


Buscador, versión escritorio (Fuente propia)

El buscador es posiblemente el elemento central de la web, por lo que su presencia es innegociable en todas las páginas de la web, si bien cada vez que se haga una búsqueda, los resultados siempre se mostrarán en la página search.

```
$searchRequest = new Request($_GET);  
$fc = New FC();  
$categories = $fc->selectCategoriesByRange();  
$all_products = [array('id_category'=> '0', 'category'=>'All Products')];  
  
array_splice($categories, 0, null, $all_products);  
$category = $searchRequest->get('category');  
$search = $searchRequest->get('terms');  
  
echo Utility::arrayToSelect('category', $categories, ['id_category','category'], true, $selected=$category);  
echo "<input id='input-text' type='text' name='terms' value='$search'>";
```

El buscador permite filtrar por categorías gracias al selector, las cuales se obtienen de forma dinámica de la base de datos. La categoría ficticia “All products” se añade al resto de categorías. Esta categoría no incluye no aplica ningún filtro de búsqueda.



The image shows a responsive search form. It contains three main elements stacked vertically: a dropdown menu with the text 'All Products' and a downward arrow, a text input field, and a button labeled 'Enviar'.

Buscador, versión responsive (Fuente propia)

Para mantener un diseño responsive, los elementos del buscador están formados como un flexbox, que permite alinear los elementos tanto en horizontal y vertical. En este caso, cuando el navegador es demasiado estrecho, los elementos del buscador se apilan en forma de columna y se igualan los anchos.

Anime	Videogames
Vocaloid Miku	
Sword Art Online	
Boku no Hero	
Attack on Titan	
Evangalion	
Kimetsu no Yaiba	
Card Captor Sakura	
Naruto	

Menú y submenú, versión escritorio (Fuente propia)

El menú muestra organiza por distintas categorías una selección de franquicias populares para que los aficionados puedan encontrar figuras fácilmente de sus personajes favoritos. Dado que las búsquedas se hacen mediante el método GET, los enlaces son en esencia redirecciones a la página del buscador con parámetros de búsqueda añadidos.

Movies / TV
Comic
Avengers
Iron Man
Spider-Man
Batman
Thor
Deadpool
Captain America
Anime
Videogames

Menú y submenú, versión responsive (Fuente propia)

Al igual que con el buscador, el menú también es un flexbox que cambia de orientación para mantener un diseño responsive.. Además, tanto el menú como el submenú igualan sus dimensiones para crear un efecto de apilación a la hora de consultar cada listado.

```
#page{
    width: 85vw;
    margin: 0 auto;
    min-height: 100vh
}
```

Finalmente, a continuación del contenedor principal hay un footer que sirve de cierre. Como el contenedor principal tiene una altura mínima del 100% de la pantalla, el footer siempre se situará al fondo de la página, independientemente del contenido de la misma.

8.4.2 Página search

Como se ha comentado anteriormente, el buscador es el eje principal de la aplicación, y por ello vamos a empezar a explicar su funcionamiento.

```
$category = $searchRequest->get('category');
$search = $searchRequest->get('terms');
$search = mb_strtolower($search);
$search_array = explode(' ', $search);
```

Cuando se realiza una búsqueda en el buscador, esta manda una petición de tipo GET a la página search. Del select recoge la categoría y del input de tipo texto se recoge el string de los términos que se utilizarán para la búsqueda. Esta cadena se convierte en minúsculas, para que las búsquedas no sean case sensitive y mediante un explode creamos un array de palabras.

Existen varios motivos para elegir el método GET sobre el método POST en este tipo de búsquedas: Se guardan en el navegador, no contienen información sensible del usuario, permiten realizar búsquedas directamente desde la URL...

Sin embargo, el principal motivo por el que se ha elegido este método en este caso es para simplificar la paginación de los resultados. Con una base de datos con cerca de 7000 productos, podemos esperar que haya búsquedas con muchos resultados, lo que implique en una sobrecarga excesiva de la página, lastrando la experiencia final del usuario.

```
if (isset($_GET['p']) && $_GET['p']!="") {  
    $pageNum = $_GET['p'];  
} else {  
    $pageNum = 1;  
}
```

La búsqueda de productos y su paginación son dos procesos que irán de la mano, así que antes de buscar, determinaremos en que número de página nos encontramos en este momento, información que sacaremos directamente de la URL mediante la superglobal \$_GET. Si no encontramos ninguna información sobre la página (En este caso, parámetro “p”), se seteará por defecto la página 1.

```
$resultsPerPage = 30;  
$offset = ($pageNum-1) * $resultsPerPage;  
$prevPage = $pageNum - 1;  
$nextPage = $pageNum + 1;
```

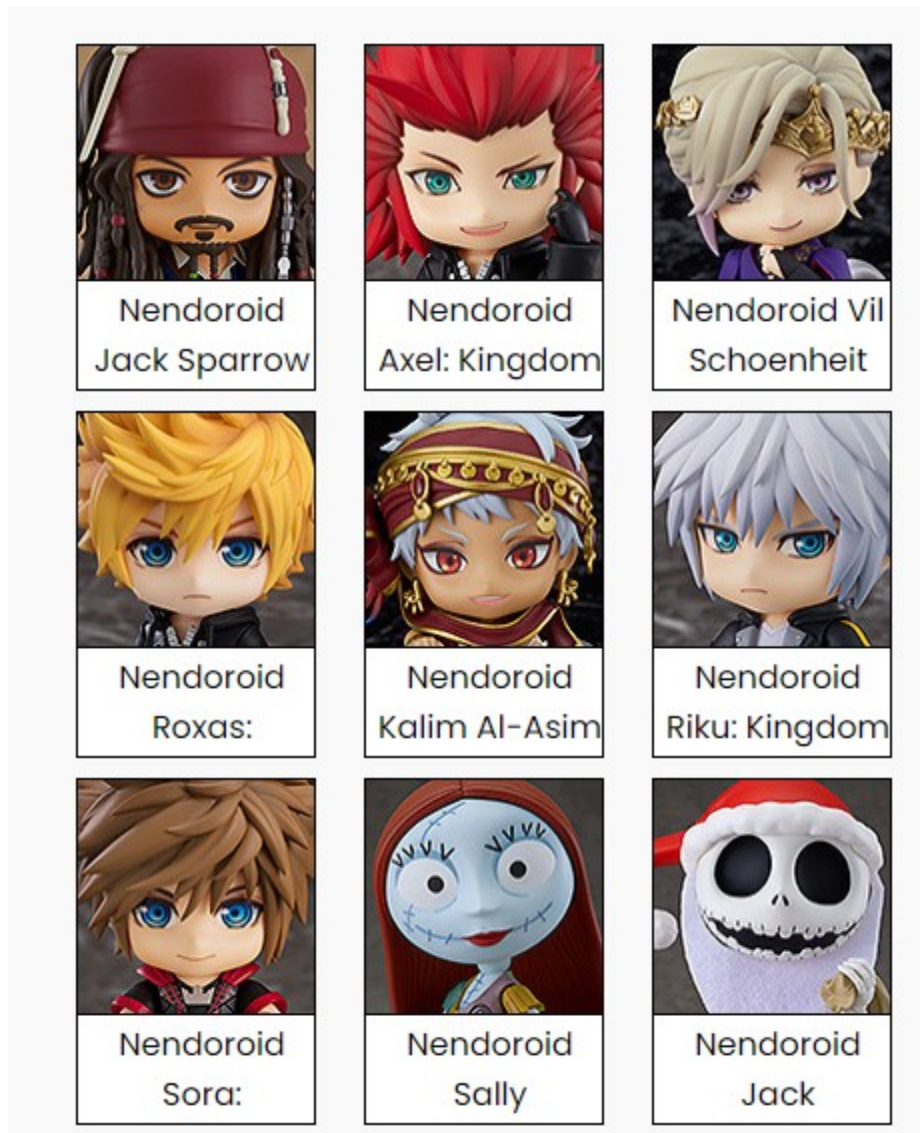
A continuación, determinaremos el límite de productos que se mostrarán por página, en nuestro caso, 30. Luego, calcularemos el offset, que determinará cuantos productos se han mostrado desde la primera página hasta la anterior a la actual. Siguiendo nuestro caso, el offset de la segunda página sería 30, en la segunda 60...

```
$result = $fc->selectProductsByTags($search_array, $offset, $resultsPerPage, $category);  
$totalResults = $fc->countProductsByTags($search_array, $category);  
$totalPages = ceil($totalResults / $resultsPerPage);
```

Lo siguiente será realizar dos queries en la base de datos: Una que calcule el total de productos que tendrá la búsqueda entre todas sus páginas, y otra que muestre la información de todos los productos de la página actual. Con el total de resultados, calcularemos cuantas páginas requerirá nuestra búsqueda, y el resultado siempre se redondeará al alza y será un entero.

```
echo '<div id="results">';  
foreach ($result as $r){  
    echo "<div class='product-container'>  
        <a href=product.php?id=\".$r['id'].\">  
            <div class='product-img'><img src='http://\".  
$r['thumbnail'].\"'></div>  
            <div class='product-name'>\".$r['name'].\"</div>  
        </a>  
    </div>";  
}  
echo '</div>';
```

Una vez tengamos los resultados, procederemos a imprimirlos en pantalla. Cada resultado se mostrará en una pequeña caja que contenga un thumbnail, el nombre del producto y un enlace a la página del mismo. Nuevamente utilizaremos flexbox para agrupar los resultados, por lo que aseguraremos un diseño responsive independientemente del tamaño de los resultados.



Ejemplo de resultados, versión responsive (Fuente propia)

```

if ($totalPages > 1){
    echo '<ul id="pagination"><li ' ;
    if ($pageNum <= 1){
        echo "class='disabled'><a";
    }
    echo '><a ' ;
    if($pageNum > 1){
        echo "href='?category=$category&terms=$search&p=$prevPage'";
    }
    echo '>&#10094;</a></li>';
}

```

Lo siguiente será crear la paginación de la página, solo en caso de que tengamos más de una página que mostrar. Lo normal es una paginación es que existan dos botones que salten a la siguiente página y a la anterior. Estos elementos solo estarán disponibles siempre que no estemos en la última página o en la primera respectivamente. Cada botón contendrá un enlace formado por los mismos elementos que el enlace que da origen a la búsqueda, pero sumando o restando el número de página.

```
echo '<li><select onchange="location=this.value;">';
for ($i=1; $i <= $totalPages; $i++){
    if ($i == $pageNum){
        echo "<option value='?category=$category&terms=$search&p=$i' selected>";
    } else {
        echo "<option value='?category=$category&terms=$search&p=$i'>$i</";
    }
}
echo '</select></li>';
```

Cuando hay una gran cantidad de páginas, existen diversas maneras de presentar la paginación. En nuestro caso, hemos apostado por un selector cuyos números contengan los enlaces a las siguientes páginas, de forma que el usuario siempre podrá navegar a una página concreta, independientemente de la página en la que se encuentre. Para que se mantenga el número de la página en el selector cada vez que se navegue a una página nueva, se utilizará una función onchange de JavaScript.



Paginador (Fuente propia)

8.4.3 Página index

Ahora que hemos visto la página search, será muy sencillo comprender el funcionamiento del index. En esta portada se hará un pequeño resumen de las últimas novedades en las 3 líneas de figuras más importantes de la marca: Nendoroid, Figma y Scale Figures.

```
$main_categories = $fc->selectCategoriesByRange(3);
echo '<div id="results">';
foreach ($main_categories as $c){
    $id = $c['id_category'];
    $cat = $c['category'];

    echo '<h1>New '.$cat.' announced!</h1>';
    echo '<div>';

    $result = $fc->selectProductsByCategoryAndSubcategory($id, $subcategories[
$i], 10);
```

Para estos casos, elegiremos las 3 categorías de la base de datos y por cada una de ellas haremos una búsqueda de un máximo de diez. También elegiremos, para cada caso, una subcategoría concreta, para obtener los resultados más. Posteriormente se paginarán los resultados siguiendo el mismo formato utilizado en search,



Página index (Fuente propia)


8.4.4 Página producto

Huelga decir que hacer una página personalizada para cada uno de los productos es una tarea irreal y hasta cierto punto ilógica, ya que la idea será mostrar todos los productos bajo un mismo formato. Para simplificar el acceso a la información de cada producto, utilizaremos su id en la URL de cada producto nuevamente mediante el método GET.

```
$fc = New FC();
$request = new Request($_GET);
if ($request->validate('id')){
    $product = $fc->selectProductById($_GET['id']);
    if (empty($product)){
        header("Location: ./index.php");
    }
    $name = $product['name'];
    echo "<title>$name</title>";
}
```

Lo primero que haremos en la página del producto es verificar que tenemos una id almacenada en la base de datos que coincida con la de la petición. En caso de que la id no sea correcta, la página nos redirigirá directamente al index. Si por el contrario todo es correcto, obtendremos el nombre del producto y lo utilizaremos como título de la página.

Nendoroid Sora: Kingdom Hearts II Ver. 1487



"The door to light... We'll go together."

From "Kingdom Hearts II" comes a new Nendoroid of Sora! The Nendoroid is fully articulated and features adjusted proportions to show his growth since the first game in Nendoroid form. He comes with three face plates including a smiling expression, a determined expression and a face to display him eating ice cream. He comes with the Kingdom Key as an optional part, allowing you to recreate battle scenes from the game. Be sure to add him to your collection!

Set Contents:
 Back and Front Hair Parts, Face Plates x3, Body, Right Arm Part xl, Right Hand Parts x3, Left Arm Part xl, Left Hand Parts x3, Crossed Arms Part xl, Right Leg Part xl, Left Leg Part xl, Kingdom Key xl

Details	
Category	Nendoroid
Series	Kingdom Hearts II
Price	¥5,400
Release Date	2021/05
Sculptor	toytec D.T.C

Página de producto, versión escritorio (Fuente propia)

Una vez que hemos pasado el primer filtro y tenemos la certeza de que el producto es correcto, seguiremos desplegando la información del producto en la pantalla. A nivel de CSS, la versión de escritorio se dividirá en dos bloques: En el lado izquierdo irá el slider de las imágenes y en el derecho se aportará el resto de información del producto: Descripción, línea, serie, precio, fecha de lanzamiento, etc...

```
function showSlides(n) {
  var i;
  var slides = document.getElementsByClassName("slider_img");
  if (n > slides.length) {slideIndex = 1}
  if (n < 1) {slideIndex = slides.length}
  for (i = 0; i < slides.length; i++) {
    slides[i].style.display = "none";
  }
  slides[slideIndex-1].style.display = "block";
}
```

El slider de imágenes se realizará mediante JavaScript. Primero se verificarán cuantas imágenes contiene el slider, para posteriormente setear todos los display de las imágenes en "none", lo que no solo hará las imágenes invisibles para el usuario, sino que impedirá que

influyan en el flujo del código HTML. Tras el bucle, se hará visible el índice actual sobre el que está situado el slider.

```
if (isset($_SESSION['user'])){\n    $user_id = $_SESSION['user']['id'];\n    $users = new Users();\n    $product_found = $users->selectProductByUserAndProduct($user_id, $id);\n    if ($product_found){\n        // Remove\n    } else{\n        // Add\n    }\n}
```

Además de toda la información restante, la página detectará si existe una sesión iniciada. En caso de que exista una sesión iniciada, verificará si un usuario posee un producto. En caso afirmativo, mostrará un botón para eliminarlo, mientras que si no existe, aparecerá un botón para añadirlo. Si no existe sesión, no mostrará nada. En la sección de Profile se hablará en más profundidad del funcionamiento de dichos botones.

```
@media screen and (max-height:900px){\n\n    #product{\n        display: flex;\n        flex-flow: column wrap;\n        align-items: center;\n        justify-content: center;\n    }\n\n    .slider_img img{\n        max-height: 700px;\n        max-width: 800px;\n    }\n\n    #slider {\n        width: 75%;\n        margin-bottom: 1em;\n    }\n}
```

Nuevamente, tendremos de una versión responsive de la página. En esta ocasión, no solo se adaptan el tamaño de las imágenes y la disposición de los elementos en función del ancho. Mediante el uso de Media Queries, detectaremos también si el navegador no está a pantalla completa, por lo que nos aseguraremos que todos los elementos se muestran de forma responsive independientemente del tamaño de la ventana del navegador.

Nendoroid Sora: Kingdom Hearts II Ver. 1487



"The door to light... We'll go together."

From "Kingdom Hearts II" comes a new Nendoroid of Sora! The Nendoroid is fully articulated and features adjusted proportions to show his growth since the first game in Nendoroid form. He comes with three face plates including a smiling expression, a determined

Página de producto, versión responsive (Fuente propia)

8.4.5 Páginas de login y alta de usuario.

Las funciones de login y alta de usuario siguen los estándares básicos de este tipo de funciones. Tanto la creación de usuario como el inicio y cierre de sesión están controlados por el archivo session.php, que cumple la función de controlador y coordina estas tareas según las peticiones que recibe por POST.

```
$location = 'Location: ./login.php';
$req_user = $req->get('user');
$req_pass = $req->get('pass');

$users = new Users();
$user = $users->selectUser($req_user);

if ($user){
    $req_pass = sha1($req_pass);
    $pass = $user['pass'];

    if ($pass == $req_pass){
        $_SESSION['user'] = $user;
        $location = 'Location: ./profile.php';
    }
}
```

En el caso del login, recoge del POST el usuario y la contraseña enviadas. Primero verifica si el usuario está registrado en la base de datos y, si es así, codifica la contraseña en sha1 y la compara con la que está asociada a ese usuario. Si todo es correcto, se inicia una sesión con los datos del usuario y se redirige al perfil del usuario. También recoge los cierres de sesión, eliminando toda la información de la superglobal \$_SESSION.

```

$new_user = $req->get('user');
$new_mail = $req->get('mail');
$new_pass = $req->get('pass');
$new_pass2 = $req->get('pass2');

if ($new_pass == $new_pass2){
    $new_pass = sha1($new_pass);

    $users = new Users();
    $user = $users->selectUser($new_user);

    if (!$user){
        $users->insertUser($new_user,$new_mail,$new_pass);
        $user = $users->selectUser($new_user);
        $_SESSION['user'] = $user;
        $location = 'Location: ./profile.php';
    }
}

```

En el caso de la creación de usuarios, se sigue un procedimiento similar. Primero se recogen todos los campos rellenados en el post y se comparan las 2 contraseñas introducidas. Si son la misma, la contraseña se codifica y se verifica que el usuario no está dado de alta en la base de datos. Si el usuario no existe, se procede a la grabación del usuario y se procede directamente al inicio de sesión.

Cabe destacar que todas las páginas del proyecto controlan si una sesión está activa. No solo se hace para mostrar la sesión iniciada del usuario en la bienvenida superior, sino también para impedir situaciones anómalas: No tiene sentido que un usuario con sesión iniciada intente hacer login o crearse una cuenta, y por supuesto ningún usuario debe poder acceder a su perfil sin iniciar sesión.

8.4.6 Página perfil.

Finalmente vamos a detallar la página de perfil del usuario. Esta página se divide en dos grandes bloques: Por un lado, muestra todas las figuras que el usuario ha marcado que tiene.

Por otro, en base a esta información, se muestra información al usuario que pueda ser de su agrado.

```
$req_type = $req->get('type');
$user = $_SESSION['user'];

if ($user){
    $id_user = $user['id'];
    $id_product = $req->get('id');
    $users = new Users();

    if ($req_type == 'add'){
        $users->insertProductUser($id_user, $id_product);
    } else if ($req_type == 'remove'){
        $users->deleteProductUser($id_user, $id_product);
    }
}
```

La gestión de la colección personal se realiza en el fichero “user_products.php”, al cual se dirigen las peticiones que realizan los botones de añadir y quitar mencionados anteriormente en la página de producto. Este archivo verifica si el usuario se ha logeado y recoge la id del producto y el tipo de petición enviada por post. En caso de que se envíe una petición de tipo “add”, se añadirá la id del producto junto a la id del usuario. Si por el contrario es una petición de tipo “remove”, se eliminará el registro combinado de usuario y producto de la base de datos.

```
$user_ids = $users->selectProductByUser($user_id);
if ($user_ids){
    $series_suggs = $fc->selectSeriesById($user_ids);
    if ($series_suggs){
        $products_suggs = $fc->selectProductsBySeries($series_suggs, $user_ids);
    }
}
```

Los productos marcados por el usuario no solo se van a utilizar para mostrarlos en la aplicación. También se van a tener en cuenta para realizar las sugerencias. En primer lugar, se obtendrán a que series pertenecen, ya que en base a esas series se realizarán las búsquedas de las sugerencias. Además, en esas búsquedas se pasarán también las ids de los productos de los que ya dispone el usuario, pero en este caso para que no los muestre, de forma que en las sugerencias solo aparecerán productos que el usuario no ha marcado y no saldrán repetidos.

Al igual que pasaba en el index, tanto los productos seleccionados como las sugerencias se mostrarán según el formato utilizado en las búsquedas.

koto figures



Suggestions for koto



Página profile (Fuente propia)

9. Fase de pruebas

En la fase de pruebas, los principales problemas surgieron cuando hubo que integrar un sistema de sesiones y usuarios a la aplicación.

Debido a la incertidumbre sobre si crear un script que hiciese las funciones de controlador (session.php), por error se estaba realizando la importación (include en PHP) de dicho script en las páginas de login y sign up.

Esto provoco un desajuste a la hora de realizar `start_session`, ya que al ejecutarse el programa, la sesión se iniciaba por duplicado: Una vez en el script y otra por la importación del archivo sesión.

El resultado era que las sesiones se sobrescribían unas a otras y los logins se perdían durante la navegación, haciendo imposible mantener una sesión iniciada correctamente al cambiar de página.

Bastó con eliminar las importaciones para corregir la incidencia.

10. Conclusiones y trabajos futuros o posibles mejoras

El resultado es una aplicación con un buscador por encima de la media que le puede situar como una aplicación referente. La generación de tags es un punto distintivo sobre las webs más punteras en el mundillo, que suelen contar con grandes bases de datos actualizadas, pero con sistemas de búsqueda muy pobres.

Posibles mejoras:

- Crear la aplicación en algo más que un gestor de colecciones y se convierta en un buscador de figuras profesional, que permita buscar y comparar precios en las distintas tiendas online especializadas.
- Toda la información del site scrapeado está en inglés, por lo que toda la aplicación en consecuencia también está en inglés. Se podría realizar, con un traductor adecuado, una versión en castellano.
- Crear un Dark Mode, para aquellas personas que sean sensibles a los colores claros.
- Normalizar completamente la base de datos, para potenciar su rendimiento, ya que la información de subcategories y series se han quedado pendientes de trasladar a su propia tabla. Esto no se ha realizado así en el proyecto dado el volumen tan inmenso de datos, que requeriría un normalizador dedicado y con una nueva actualización para cada nuevo producto que saliese al mercado.
- Con una base de datos normalizada, se podrían verificar que series son las que tienen más productos y situarlas en el menú de forma dinámica.
- Mejorar la exclusión de palabras incluidas en las tags.
- Añadir más opciones a la gestión del usuario: Envío de confirmaciones por correo, posibilidad de cambiar la contraseña, añadir un avatar, sistema de amigos con otros usuarios, etc...

- Detección de errores más eficaz
- Uso de AJAX y APIs para las peticiones.
- Uso de sistemas de login más seguros.

11. Referencias bibliográficas

W3Schools - <https://www.w3schools.com/>

stack overflow - <https://stackoverflow.com/>

php.net – <https://www.php.net>