

MEMORIA PRACTICA 2

Esta práctica ha sido realizada por:

- Sofía Sánchez: sofia.sanchezf@estudiante.uam.es
- Victoria Pelayo: victoria.pelayo@estudiante.uam.es

Descripción de los archivos de entrega

A continuación explicaremos los directorios que contiene nuestra práctica.

- app: En esta carpeta se encuentran todos los archivos necesarios para que la web implementada funcione. Además, dentro de esta encontramos diferentes carpetas que también explicaremos que contienen.
 - o catalogue: Contiene el fichero *catalogue.json* que contiene la información de nuestro videoclub, es decir, información de las películas que se encuentran disponibles.
 - o static: Contiene otra carpeta llamada *images*, donde encontramos todas las imágenes necesarias para nuestra página web, y el archivo *style.css*.
 - o templates: Contiene todos los archivos *.html* necesarios para la web implementada. En nuestro caso hemos necesitado:
 - base.html: este es el archivo principal, a través de este el cliente podrá acceder al sistema. El resto de html's exiten de este.
 - index.html: Contiene el código HTML correspondiente a la página principal, es decir, nos muestra las películas en la zona de contenidos.
 - carrito.html: Contiene el código HTML correspondiente a la información del carrito, es decir, nos muestra en la zona de contenido una tabla con el contenido del carrito de la compra.
 - historial.html: Contiene el código HTML correspondiente a la información del historial de compra de un usuario; dicha información se mostrará en una tabla.
 - informacion.html: Contiene el código HTML correspondiente a la información detallada de una película.
 - login.html: Contiene el código HTML que permite a un cliente logearse en la web.
 - registro.html: Contiene el código HTML que permite al cliente registrarse en la página web.
 - o thesessions: se encuentran los datos de las sesiones de la web
 - o usuarios: Contiene una carpeta para cada usuario registrado en la

aplicación. Estas carpetas contienen a su vez los ficheros *datos.dat* e *historial.json*.

También contiene el *routes.py* (donde tendremos diferentes funciones que explicaremos a lo largo de la memoria), *__main__.py* y *__init__.py*.

Estructura de la web

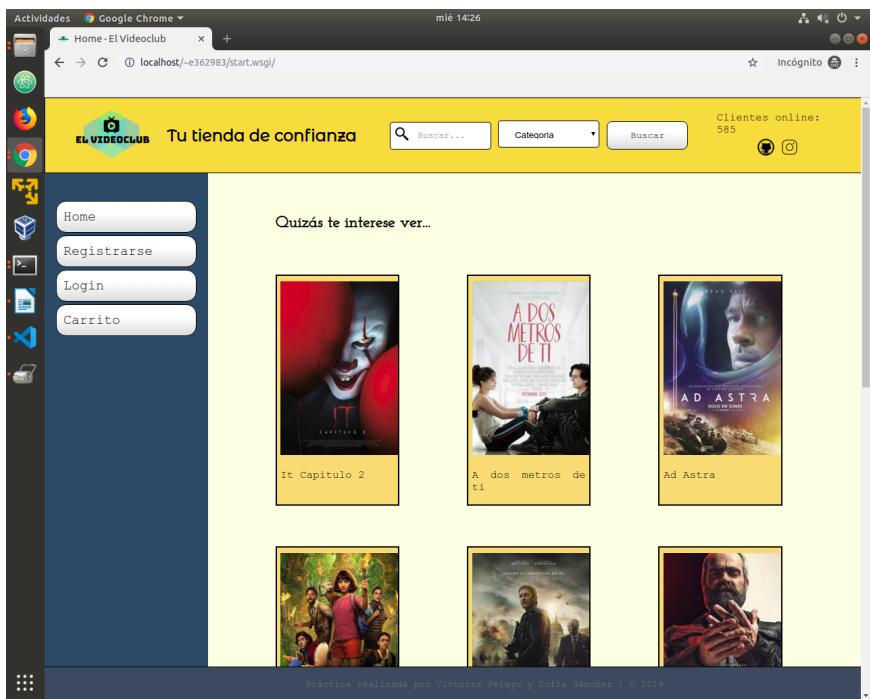
Página principal

Nuestra página web, al igual que en la práctica anterior, cuenta con una cabecera fija, un pie de página fijo, un menú lateral y una zona para mostrar los contenidos. Todo el código HTML correspondiente a esto se encuentra en el archivo *base.html* del cual extienden todos, pues son los elementos comunes para el resto de archivos *.html*.

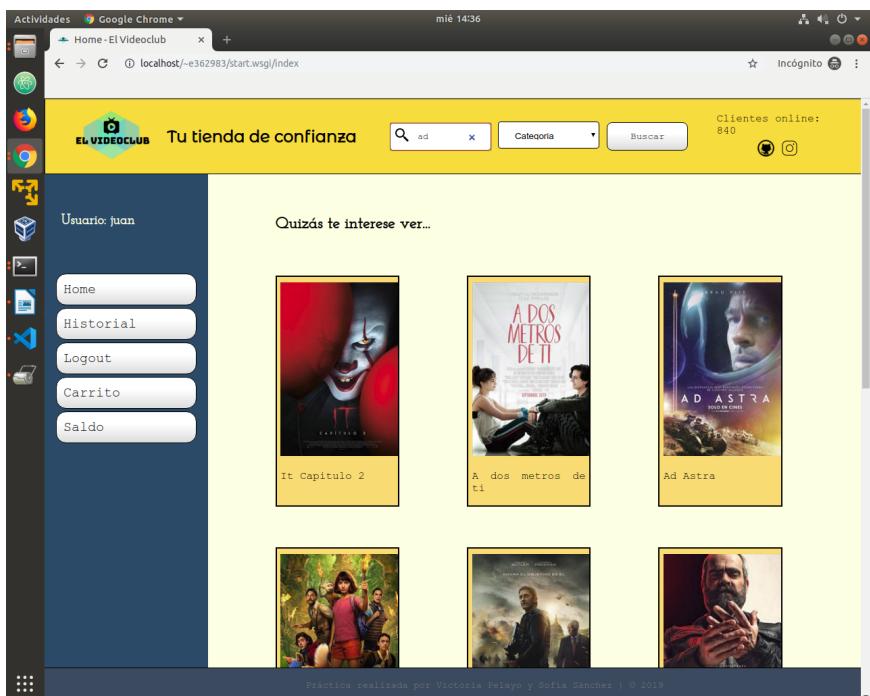
En cuanto al menú lateral, dependiendo de si el usuario está registrado o no será diferente puesto que las funcionalidades que tiene el usuario registrado y el que no son distintas. Respecto a la cabecera, es siempre igual y cuenta con el logo de la aplicación en el cual si pinchamos volveremos de nuevo a la página principal (tanto usuarios registrados como no registrados), cuenta con una zona de búsqueda formada por un desplegable, un *input* para escribir el título de la película a buscar y un botón y por último cuenta con una zona de información donde se muestran los usuarios conectados al sistema y dos métodos de contacto con los propietarios de la web (github e Instagram).

Para implementar la funcionalidad de mostrar los usuarios conectados al sistema utilizaremos AJAX, tal y como se nos pide en el enunciado. Para ello hemos creado en *base.html* la función *ajaxd()* que envía peticiones tipo GET al servidor para obtener el número de usuarios.

La página principal se encuentra en *index.html*, que como ya hemos dicho extiende de *base.html*. Para cargar esta página hemos creado la función *index()* en *routes.py* que se encargará de crear el carrito en la sesión si no hay uno ya creado de tal manera que creamos el *carrito* como una lista vacía, ponemos el *precio* total de la compra a 0 y creamos una lista llamada *n_producto_carrito* que contendrá la cantidad de veces que hemos añadido cada película al carrito. Tras esto, indicaremos que la sesión ha sido modificada y cargaremos *index.html* pasándole una lista de las 10 primeras películas que se encuentran en el *catalogo.json*.



La página principal para un usuario registrado se vera de la siguiente manera:



En cuanto a la funcionalidad para **buscar películas** en la aplicación debemos saber que nos aparecerán las películas que contienen la palabra escrita en ese cuadro de texto. Además, tendremos dos opciones: buscar por título y categoría o buscar solo por título.

Para buscar la película hemos implementado la función `busqueda()` en `routes.py`. En dicha rutina crearemos una lista con todas las películas del `catalogo.json` y otra lista, llamada `L`, que estará vacía inicialmente.

Lo primero que haremos comprobar si el campo buscar está vacío, en caso de que no lo

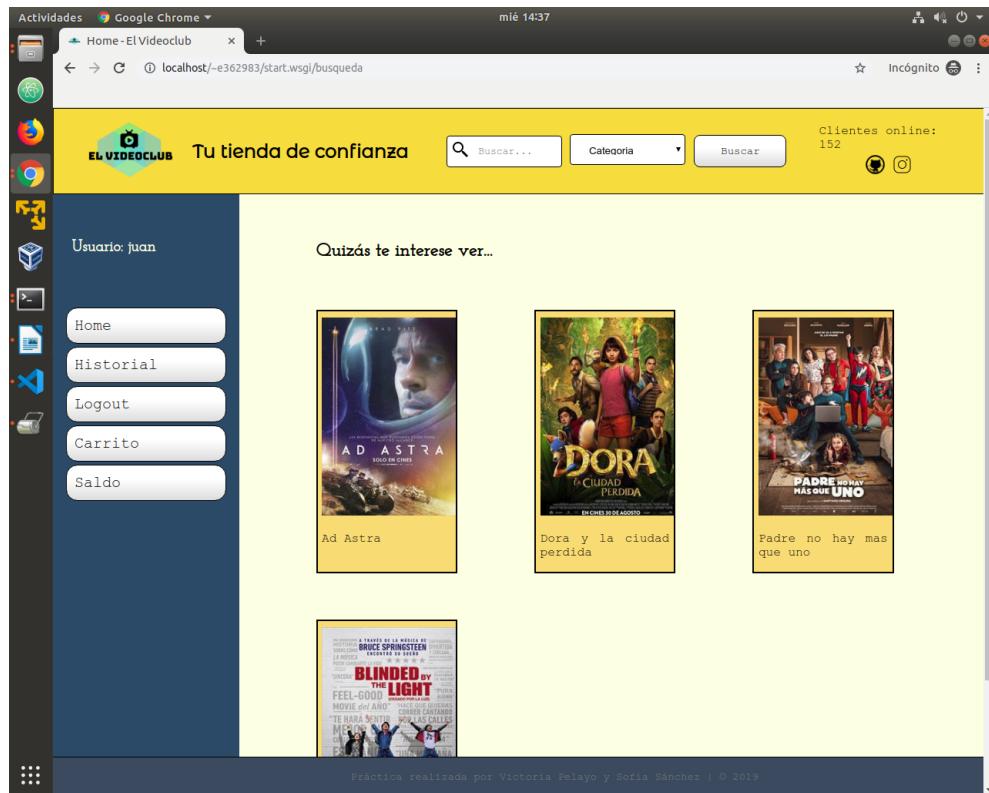
esté para cada ítem en la lista de películas del catálogo comprobaremos que contenga la palabra por la que estamos buscando y la añadiremos a la lista L. Tras esto, miraremos si tenemos categoría por la que queremos filtrar; en caso de tenerla, los ítems que no tengan la categoría requerida serán eliminados de la lista L.

En caso de que el campo buscar este vacío y tengamos que filtrar solo por categoría, comprobaremos que para cada ítem en la lista de películas del catálogo se tenga dicha categoría. Si se tiene, se añadirá a la lista L.

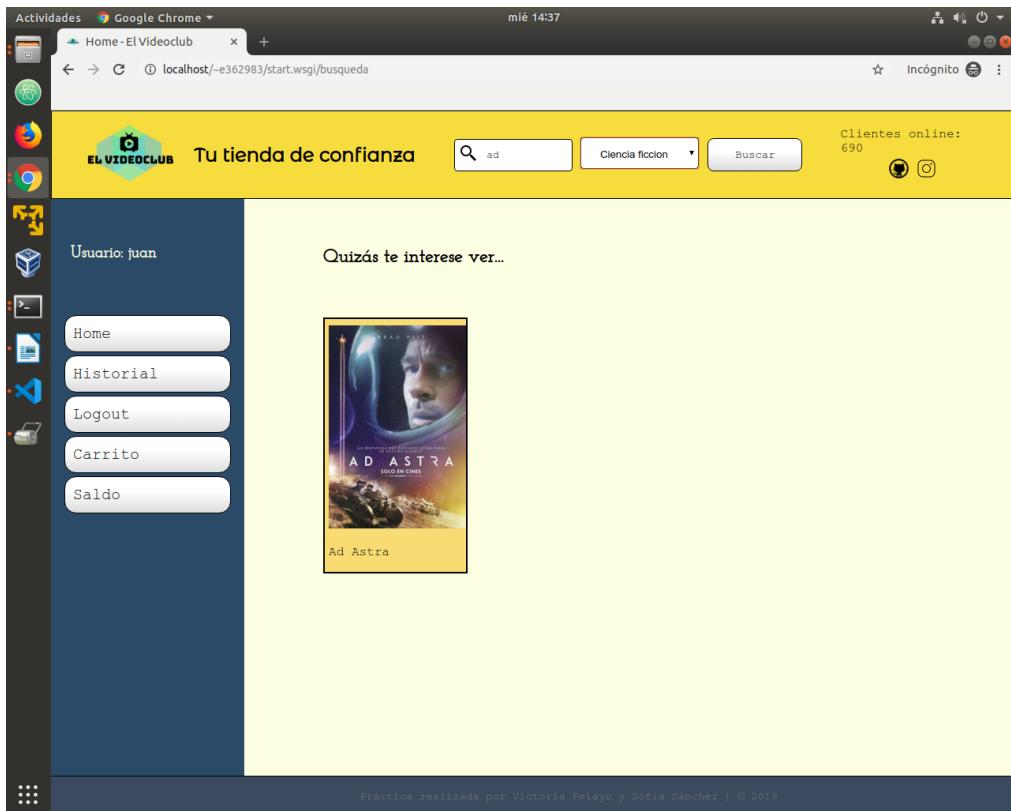
Finalmente, cargaremos *index.html* pasándole como atributo la lista L de películas que se deben mostrar.

A continuación, mostramos un ejemplo de ambos casos:

Si buscamos ad sin categoría el resultado es el siguiente:



Si buscamos ad con categoría Ciencia Ficción el resultado es el siguiente:



Acceso de un usuario

Mediante esta página un cliente podrá realizar el **login** introduciendo el nombre y su contraseña. Para ello contamos con una función en el fichero *routes.py* que nos comprueba que el usuario se encuentre registrado en la aplicación y además que la contraseña y el nombre sean los correctos. En caso de que el login no se realice correctamente, se mostrará un mensaje en la zona de los contenidos.

Al igual que en la práctica anterior, hemos hecho uso de *form* para el formulario. Para poder iniciar sesión en la aplicación hemos implementado la función *login()* en *routes.py*.

En la rutina citada anteriormente comprobaremos que exista *username* en la request del formulario. En caso afirmativo, obtendremos la contraseña y el nombre de usuario del fichero *datos.dat* del usuario. Tras esto, codificaremos en md5 la contraseña que nos han introducido en el formulario y comprobaremos ambos campos (nombre de usuario y contraseña) con los que hemos obtenido del fichero de datos del usuario. En caso de que coincidan, introduciremos un usuario en la sesión, guardaremos una cookie con el nombre del ultimo usuario en realizar login (lo haremos mediante las funciones de flask *make_response* y *set_cookie*), indicaremos que la sesión ha sido modificada y redirigiremos al *index.html* mediante la función *redirect*. En caso de que los datos introducidos no sean los correctos, mostraremos un mensaje de error que nos indica que el login no se ha realizado correctamente.

Si no existe `username` en la request, lo que haremos será recuperar la cookie que habíamos guardado con el nombre del ultimo usuario que realizó login mediante la función de `request.cookies.get()` e introduciremos este valor en el campo `username` del formulario para que quede autocompletado.

A continuación, mostramos capturas de lo que hemos explicado

Actividades Google Chrome ▾

milé 14:32

Login - El Videoclub x +

localhost/-e362983/start.wsgi/login

Tu tienda de confianza Clientes online: 898

EL VIDEOCLUB

Buscar... Categoría Buscar

Home Registrarse Login Carrito

Inicia sesión!

Nombre de Usuario: Completa este campo

Contraseña:

Confirmar

¿Aún no estás registrado?

Registrate

Práctica realizada por Victoria Pelayo y Sofía Sánchez | © 2019

En el caso de que juan haya sido el ultimo usuario en hacer login, tendremos lo siguiente:

Actividades Google Chrome ▾

milé 14:37

Login - El Videoclub x +

localhost/-e362983/start.wsgi/login

Tu tienda de confianza Clientes online: 910

EL VIDEOCLUB

Buscar... Categoría Buscar

Home Registrarse Login Carrito

Inicia sesión!

Nombre de Usuario: juan

Contraseña:

Confirmar

¿Aún no estás registrado?

Registrate

Práctica realizada por Victoria Pelayo y Sofía Sánchez | © 2019

Logout de un usuario

Esta funcionalidad la hemos implementado mediante la función *logout()* en *routes.py* que se encarga de eliminar de la sesión al usuario mediante un *pop* y redirigiéndonos a la página principal.

Registro de un usuario

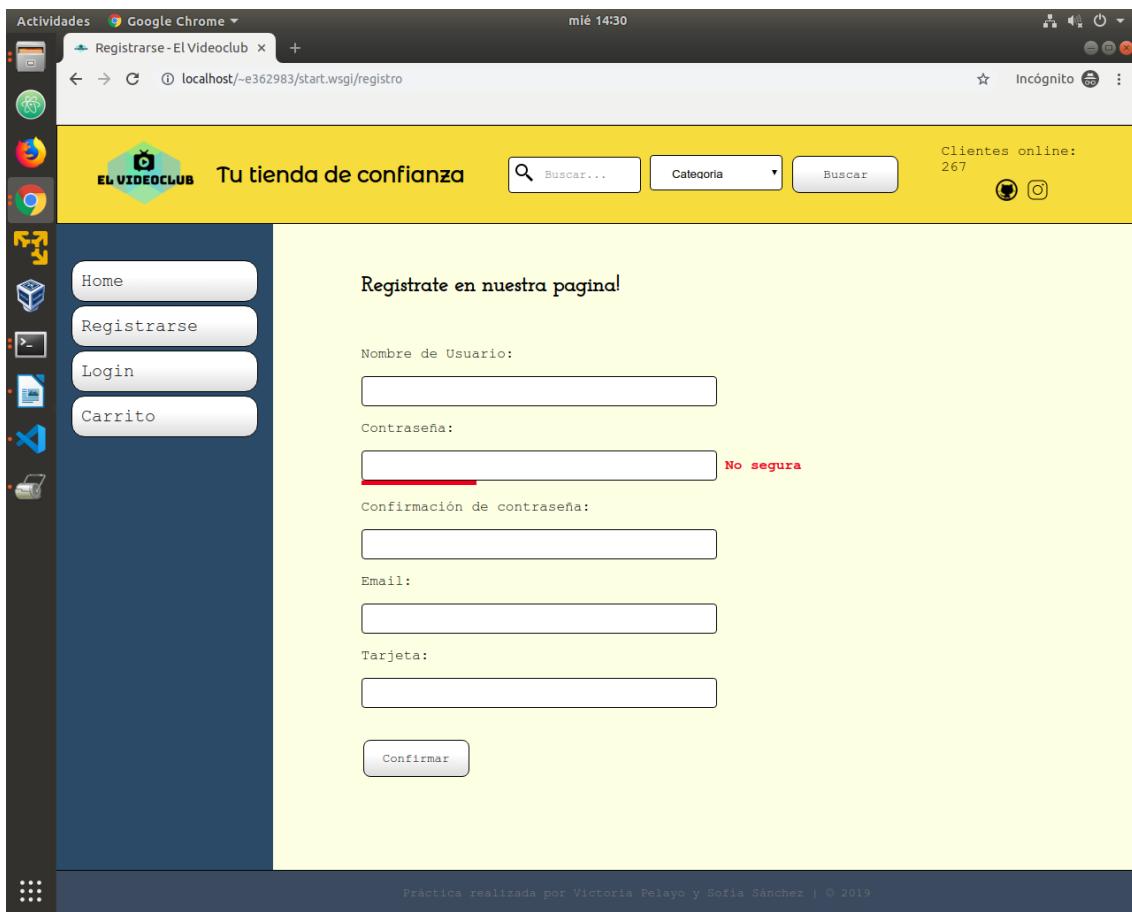
Esta es la página a través de la cual un usuario puede registrarse en la web. La hemos implementado usando *form* para el formulario, explicado anteriormente en la práctica 1. El código HTML correspondiente a la página del registro de un usuario se encuentra en *registro.html* que extiende de *base.html*.

En esta práctica hemos tenido que **comprobar que los datos introducidos en el formulario** eran correctos para ello hemos hecho uso de *JavaScript* creando una función llamada *validar()*. Dicha rutina contiene la variable *todo_correcto* que comienza siendo *True*, después realizamos las comprobaciones pertinentes: el nombre introducido tiene más de 2 caracteres, el nombre no contiene caracteres no permitidos, se ha introducido una contraseña de más de 8 caracteres, se ha introducido una dirección de correo válida y la tarjeta tiene 12 dígitos. Si no se cumplen algunas de las condiciones se mostrará un mensaje de alerta mediante la función *alert()* de *JavaScript* y además pondremos *todo_correcto* a *False*. En caso de que se cumplan todas las condiciones devolveremos *True*.

De esto cabe destacar tres cosas: para la comprobación de los caracteres permitidos hemos hecho uso de la función *permite()* que se encarga de comprobar si contiene caracteres inválidos o no; para la comprobación de la dirección de correo hemos definido una expresión regular y posteriormente comprobaremos si en el email introducido se encuentra dicho patrón; por último, para la comprobación de la fortaleza de la contraseña hemos hecho uso de *jQuery* y para ello hemos comprobamos la longitud de la contraseña y en función de esto nos indica si es poco segura, medianamente segura o segura.

Una vez que el usuario ha introducido todos los datos correctamente llamaremos a la función *registro()* implementada en *routes.py*. A través de esta rutina comprobaremos que el nombre del usuario a registrar sea distinto de otro que se encuentre en la aplicación, en caso contrario, mostraremos un mensaje de error. Si no se encuentran coincidencias y el usuario no existe, crearemos el directorio para dicho usuario dentro de la carpeta *Usuarios* con su correspondientes archivos *datos.dat* e *historial.json*.

En *datos.dat* guardamos inicialmente el nombre de usuario, la contraseña, el correo electrónico, el número de tarjeta y el saldo (que es un número aleatorio entre 0 y 100).



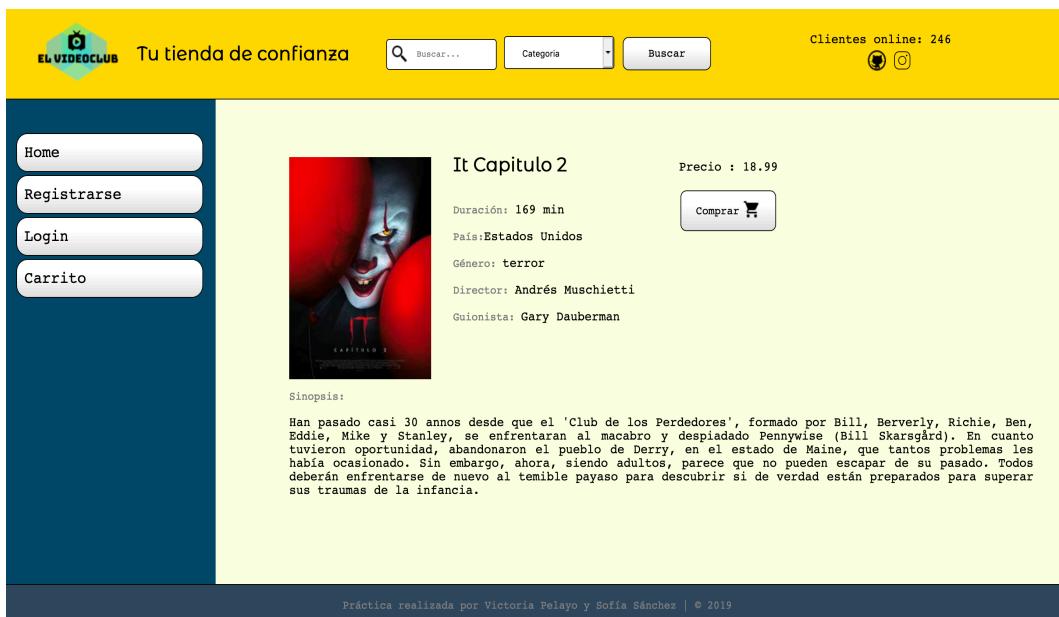
Detalle de la película

Esta es la página mediante la cual podemos ver de manera detallada una película. En este caso para mostrar la información de la película debemos de pasarle a la template la película (la pasaremos a través del atributo *film*) y obtener los datos de la siguiente manera: {{film.duracion}}, {{film.pais}}, {{film.categoría}}, {{film.director}} y {{film.guionista}}, pues son los datos que se encuentran en el fichero *catalogue.json*.

Para esta funcionalidad hemos implementado en *routes.py* la función *informacion()* la cual recibe como argumento de entrada el id de la película que queremos ver de manera detallada. En dicha rutina crearemos una lista con todas las películas del *catalogo.json* y comprobaremos si *película_id* coincide con alguno de los id's de dicha lista de películas. En caso de coincidir, cargaremos *informacion.html* pasándole la película en el atributo *film*.

Una vez que vemos la película detallada tenemos un botón que nos permite la opción de añadirla al carrito. Para esta funcionalidad hemos implementado la función *comprar()* en *routes.py*. la cual recibe como argumento de entrada el id de la película que se quiere comprar. En dicha rutina obtendremos una lista con todas las películas del *catalogo.json* y comprobaremos para cada ítem en la lista si su id coincide con el que recibimos como argumento de entrada. En caso de coincidir, añadiremos el id a la lista carrito de la sesión que encuentra actualmente e incrementaremos el precio total del

carrito.



Carrito de la compra

Esta es la página mediante la cual podemos ver el carrito de la sesión que se encuentra actualmente. Los datos se mostrarán en una tabla donde tendremos la opción de eliminar películas del carrito y finalizar la compra. El código HTML correspondiente a esta página se encuentra en *carrito.html*.

Contamos con la función *carrito()* en *routes.py* que será la función encargada de mostrarnos la template *carrito.html* donde se mostrará la lista de películas que se encuentran actualmente en el carrito de la compra.

Para eliminar las películas del carrito hemos creado la función *eliminar()* en *routes.py* la cual recibe como argumento de entrada el id de la película que queremos eliminar. En dicha rutina crearemos una lista con todas las películas del *catalogo.json* y comprobaremos para cada ítem en dicha si el id coincide con el id que recibimos como argumento. En caso de que coincida, eliminamos el id de la lista del carrito en nuestra sesión (mediante *remove*), decrementaremos el precio total del carrito en dicha sesión e indicaremos que la sesión ha sido modificada. Finalmente redirigimos al carrito mediante la función *redirect*.

Para finalizar la compra hemos creado la función *finalizar()* en *routes.py*. En dicha rutina lo primero que haremos será crear una lista con todas las películas del *catalogo.json*. Tras esto, comprobaremos que el usuario este registrado en la aplicación; en caso de que este loggeado comprobaremos que tenga saldo suficiente para hacer la compra (es decir es mayor que 0 o es mayor que el precio total de la compra). Si no tiene el saldo suficiente volveremos a mostrar *carrito.html* con un mensaje de error indicando que no se dispone del saldo suficiente. Por el contrario, si tiene el saldo suficiente, lo modificaremos en el fichero *datos.dat* del usuario y se lo restaremos, añadiremos dicha

película al historial del usuario guardado en *historial.json* y la fecha en la que se ha realizado la compra, pondremos el precio total de la sesión a 0, la lista del carrito de la sesión vacía, la lista que contiene la cantidad de cada producto la inicializaremos a 0 de nuevo e indicaremos que la sesión ha sido modificada. En caso de que el usuario no este loggeado, le llevaremos a la página de *login.html* para que inicie sesión o se registre y pueda finalizar la compra.

Película	Precio	Cantidad	
	18.99	8	<button>Eliminar</button>
	18.99	3	<button>Eliminar</button>
	18.99	5	<button>Eliminar</button>

Histórial de un usuario

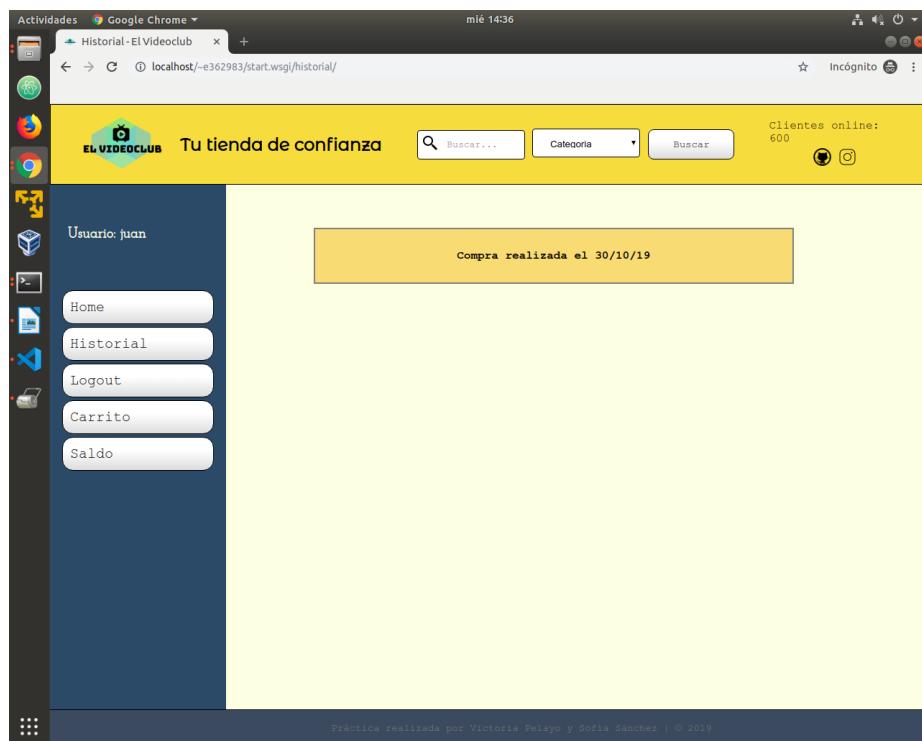
Esta es la página mediante la cual podemos mostrar el histórico de compras de un usuario (*historial.html*) y estas aparecen según la fecha en la que se realizó. Apreciamos que si pinchamos sobre la fecha se nos mostrarán detalles de la compra que realizamos en dicha fecha apareciendo así todas las películas compradas como la cantidad que se compraron.

Para implementar esta funcionalidad hemos hecho uso de **jQuery**, tal y como se pedía en el enunciado, de manera que hemos añadido *<script>* donde colocaremos las funciones de jQuery. En concreto, hemos usado la función *slideToggle()* para desplegar y contraer los detalles de la compra de manera que dicha información esté inicialmente oculta y cuando pinchamos se despliega y si volvemos a pinchar se oculta. Para poder usarla ha sido necesario importar la librería de jQuery que la contiene.

Cabe destacar qué hemos decidido guardar en el archivo *historial.json* que tiene cada

usuario. En nuestro caso, hemos decidido que se guarden como diccionarios donde la clave es la fecha en la que se realizó la compra y los valores se corresponden con las diferentes películas que se han comprado. Si una película se ha comprado 2 veces, no se guardará dos veces sino que crearemos dentro una nueva clave llamada *cantidad* que será la que nos indique cuantos ejemplares hemos comprado de dicha película.

También hemos creado la función *historial()* en *routes.py* que nos permite mostrar la página *historial.html* que recibe la lista de películas del catálogo.



Cuando pinchamos sobre la compra, nos aparecerá de la siguiente manera:

The screenshot shows a web browser window titled 'Actividades Google Chrome'. The URL is 'localhost/~e362983/start.wsg/historial/'. The page has a yellow header with the logo 'EL VIDEOCLUB' and the tagline 'Tu tienda de confianza'. It also displays 'Cuentas online: 954'. On the left, there is a sidebar for the user 'juan' with options: Home, Historial, Logout, Carrito, and Saldo. The main content area shows a table titled 'Compra realizada el 30/10/19' with three rows of data:

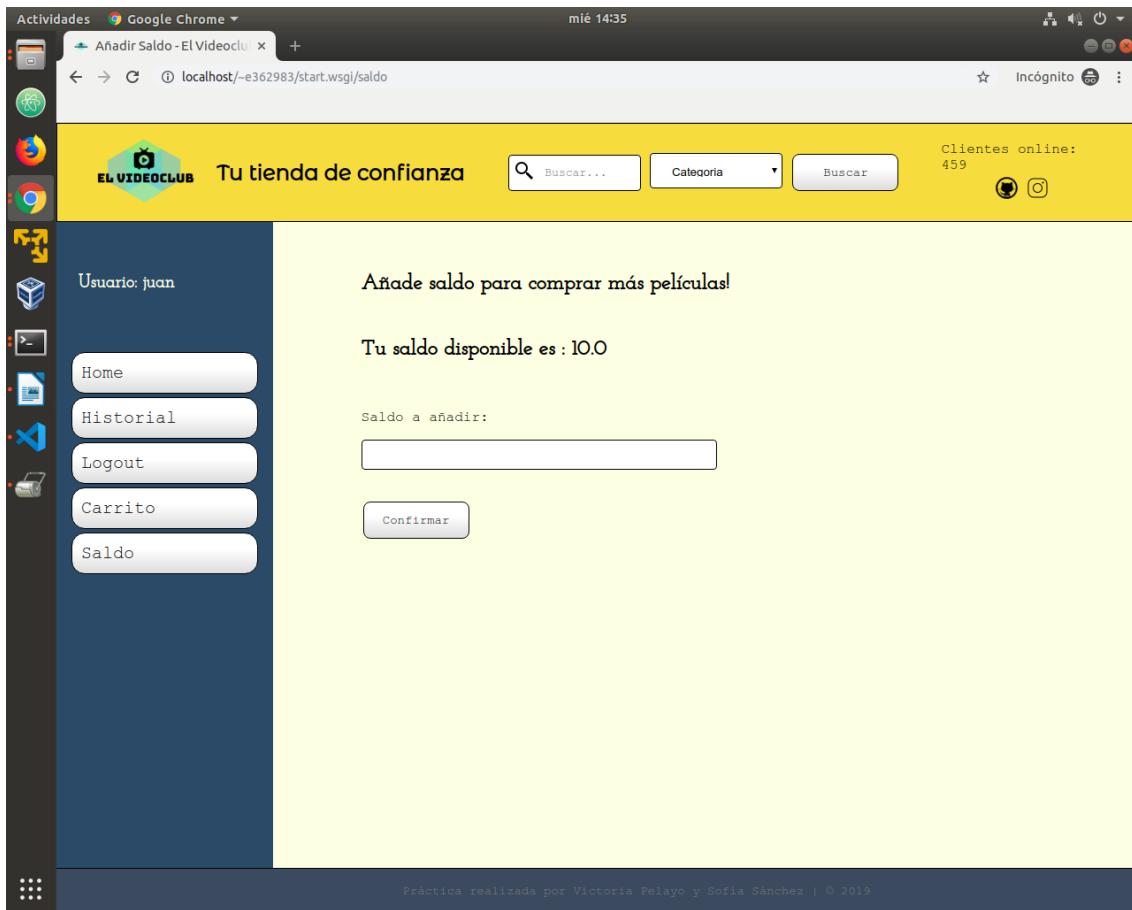
Imagen	Título	Precio	Cantidad
	A dos metros de ti	18.99	3
	It Capítulo 2	18.99	8
	Ad Astra	18.99	5

At the bottom of the page, it says 'Práctica realizada por Victoria Pelayo y Sofía Sánchez | © 2019'.

Saldo de un usuario

Esta página nos permite modificar el saldo de un usuario y su código HTML correspondiente se encuentra en *saldo.html*.

Para implementar esta funcionalidad hemos creado la función *saldo()* en *routes.py*. En esta rutina lo primero que haremos será obtener el saldo del usuario (tan solo con dos decimales) para poder mostrarlo, seguidamente comprobaremos que en el formulario se haya introducido el número que queremos sumarle al saldo. Si se ha introducido, tendremos que actualizar el fichero *datos.dat*. En nuestro caso lo que hemos hecho ha sido escribir todo el fichero de nuevo excepto la última línea donde escribiremos el saldo nuevo. En caso de que no se introduzca, no realizaremos nada y se mostrará el saldo que ya teníamos.



Funcionamiento

La práctica ha sido probada tanto de manera local como en apache.

Para probarla en apache la hemos puesto en el fichero public_html, hemos creado un entorno virtual donde hemos instalado *requirements.txt* que contiene lo siguiente:

```
Click==7.0
Flask==1.1.1
Flask-Session==0.3.1
Flask-SQLAlchemy==2.4.1
itsdangerous==1.1.0
Jinja2==2.10.1
MarkupSafe==1.1.1
psycopg2==2.8.3
six==1.12.0
SQLAlchemy==1.3.8
SQLAlchemy-Utils==0.34.2
Werkzeug==0.16.0
```

Tras esto, hemos tenido que dar permisos a las carpetas *usuarios* y *thesessions* para que se puedan crear usuarios nuevos y crear nuevas sesiones.

Finalmente, en el navegador ponemos `localhost/~usuario` y ya tenemos nuestra página web.

Mostramos una captura de pantalla del terminal con los comandos necesarios tanto para crear el entorno virtual como para dar permisos a las carpetas citadas anteriormente.

```
e362983@2-5-2-5:~/public_html$ virtualenv siipyenv
Using base prefix '/usr'
New python executable in /home/alumnos/e362983/public_html/siipyenv/bin/python3
Also creating executable in /home/alumnos/e362983/public_html/siipyenv/bin/python
Installing setuptools, pip, wheel...done.
e362983@2-5-2-5:~/public_html$ touch requirements.txt
e362983@2-5-2-5:~/public_html$ source siipyenv/bin/activate
(siipyenv) e362983@2-5-2-5:~/public_html$ pip3 install -r requirements.txt
Collecting Click==7.0
  Downloading https://files.pythonhosted.org/packages/fa/37/45185cb5abbc30d7257104c434fe0b07e5a195a6847506c074527aa599ec/Click-7.0-py2.py3-none-any.whl (81kB)
    |████████| 81kB 678KB/s
Collecting Flask==1.1.1
  Downloading https://files.pythonhosted.org/packages/9b/93/628509b8d5dc749656a9641f4caf11540e2cdec85276964ff8f43bb1d3b/Flask-1.1.1-py2.py3-none-any.whl (94kB)
    |████████| 102kB 1.7MB/s
Collecting Flask-Session==0.3.1
  Downloading https://files.pythonhosted.org/packages/7e/a3/3ba4cec2becb7c4e65df2a95052c7050832c12a1fc8a475ac572842c460bc/Flask_Session-0.3.1-py2.py3-none-any.whl
Collecting Flask SQLAlchemy==2.4.1
  Downloading https://files.pythonhosted.org/packages/1e/65/226d95466c75e34e291a76890ed0e27afe246ab913002847856f11d4d59d/Flask_SQLAlchemy-2.4.1-py2.py3-none-any.whl size=1193295 sha256=3067cb0c30e057ccc5a69baeb7c1ba1abd0077292c111314439e59499083dd0e
  Stored in directory: /home/alumnos/e362983/.cache/pip/wheels/97/b6/66/de2064d40c920adc2984ff3bf4df11494c8ab9e48ba87e8a2
  Building wheel for SQLAlchemy-Utils (setup.py) ... done
    Created wheel for SQLAlchemy-Utils: filename=SQLAlchemy_Utils-0.34.2-py2.py3-n
one-any.whl size=90094 sha256=ce2ce864831c29c610033dd067d0141f14ea7d4495f9e904c1cc6902898e421
  Stored in directory: /home/alumnos/e362983/.cache/pip/wheels/74/73/55/6d2e33ae4013e904ab1b7458a50732134b45956d923a050999
Successfully built psycopg2 SQLAlchemy SQLAlchemy-Utils
Installing collected packages: Click, MarkupSafe, Jinja2, itsdangerous, Werkzeug
, Flask, Flask-Session, SQLAlchemy, Flask-SQLAlchemy, psycopg2, six, SQLAlchemy-Utils
Successfully installed Click-7.0 Flask-1.1.1 Flask-SQLAlchemy-2.4.1 Flask-Session-0.3.1 Jinja2-2.10.1 MarkupSafe-1.1.1 SQLAlchemy-1.3.8 SQLAlchemy-Utils-0.34.2 Werkzeug-0.16.0 itsdangerous-1.1.0 psycopg2-2.8.3 six-1.12.0
(siipyenv) e362983@2-5-2-5:~/public_html$ deactivate
e362983@2-5-2-5:~/public_html$ cd app/
e362983@2-5-2-5:~/public_html/app$ setfacl -m u:www-data:rwx usuarios
setfacl: opción -m: Argumento inválido cerca del carácter 3
e362983@2-5-2-5:~/public_html/app$ setfacl -m u:www-data:rwx usuarios
e362983@2-5-2-5:~/public_html/app$ setfacl -m u:www-data:rwx thesessions
```

Si por el contrario lo queremos crear de manera local, lo que tendremos que hacer será situarnos en la carpeta en la que tenemos la app y realizar el comando `python3 -m app` y en el navegador poner localhost:5002 (puesto que el puerto que hemos elegido en el main.py es el 5002)

A continuación dejamos una captura de pantalla:

```
[~]e362983@2-5-2-5:~$ python3 -m app
Usando sesiones de Flask-Session en fichero del servidor
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
Usando sesiones de Flask-Session en fichero del servidor
 * Debugger is active!
 * Debugger PIN: 305-816-804
```