MEMORIA PRÁCTICA 2

Alejandro Santorum Varela - alejandro.santorum@estudiante.uam.es
Rafael Sánchez Sánchez - rafael.sanchezs@estudiante.uam.es
Sistemas Informáticos I
Práctica 2 Pareja 9

30 de octubre de 2019

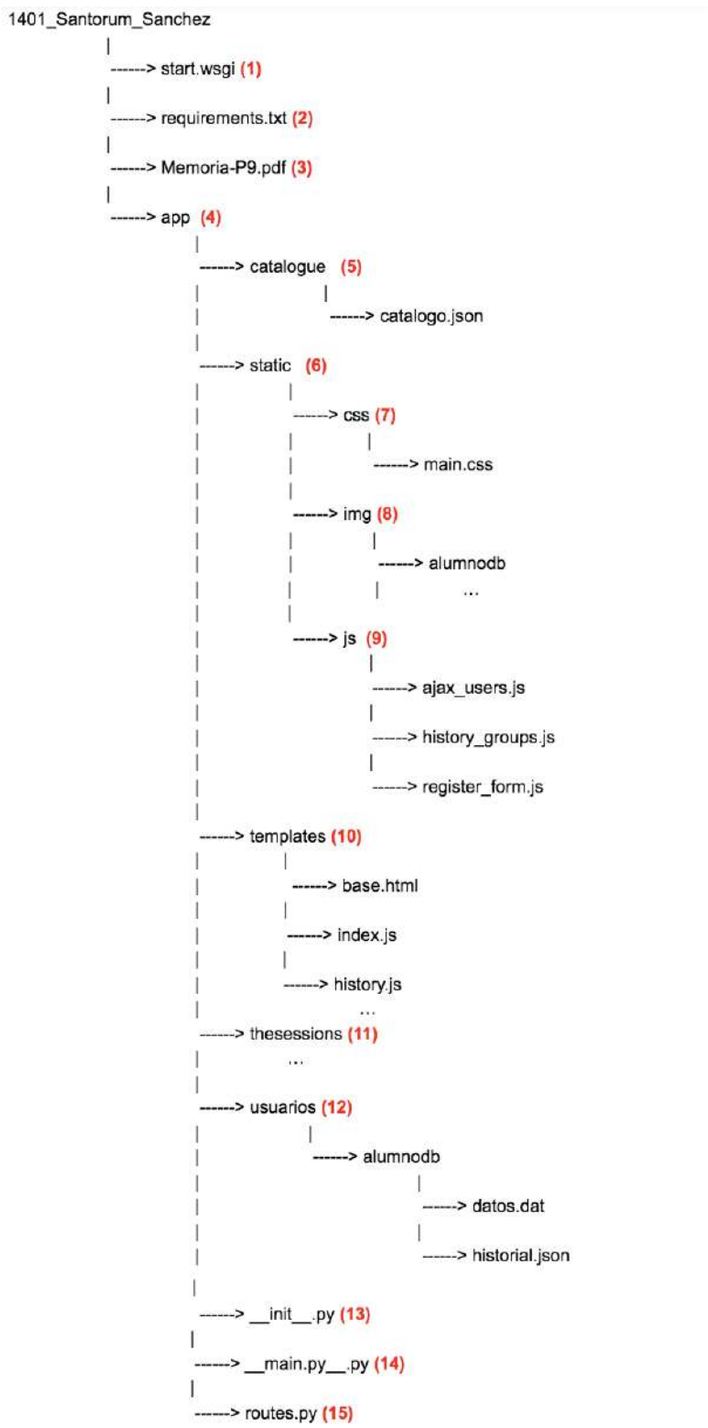
Contents

1 Introducción

Nos encontramos ahora en la segunda práctica de Sistemas Informáticos I. En esta nos centraremos en realizar el *backend* de la aplicación web (toda la lógica interna), con pequeñas modificaciones en el *frontend* desarrollado en la primera práctica.

En esta memoria se recoge la funcionalidad de nuestra página - con varias capturas de pantalla - y el proceso de ejecución que se debe seguir si queremos correr la aplicación tanto en local como bajo el servidor Apache.

2 Estructura de carpetas y ficheros de la entrega



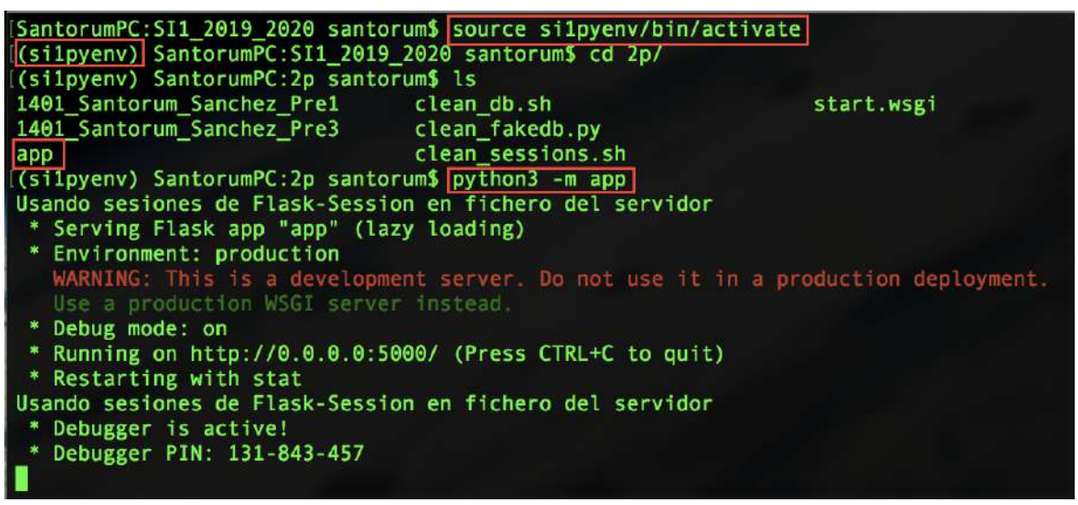
En primer lugar comentaremos qué estructura posee la carpeta de la entrega, que sigue lo estipulado de la `Normativa de prácticas`:
(1) Fichero **start.wsgi**, en la raíz, se encarga de arrancar la aplicación bajo el servidor apache.
(2) Fichero **requirements.txt**, en la raíz, usado para configurar el *virtual environment* para

instalar todos los paquetes necesarios.
(3) Fichero **Memoria-P9.pdf** este documente, la memoria de la práctica.
(4) Directorio **app**, donde se encuentra todo el código de la aplicación. Dentro de esta nos encontramos con:
(5) Directorio **catalogue**, que posee en su interior a su vez el fichero **catalogo.json**, fichero donde tenemos alguna información de películas de ejemplo, simulando nuestra base de datos de películas.
(6) Directorio **static**, donde encontraremos los ficheros estáticos de la aplicación. Dentro de este directorio podemos encontrar los subdirectorios `css`, `img` y `js`:
(7) Directorio **css**, subdirectorio de `static`, contiene los ficheros de estilo `css`, en este caso solo **main.css**.
(8) Directorio **img**, subdirectorio de `static`, contiene las imágenes de los pósters de las películas de ejemplo, y una carpeta por cada usuario que haya subido una foto de perfil propia a su cuenta. En esta carpeta se guardará la foto de perfil subida.
(9) Directorio **js**, subdirectorio de `static`, contiene los ficheros de los scripts de Javascript usados para la validación de formularios y la generación del número de usuarios conectados.
(10) Volviendo a la lista de los subdirectorios de `app`, directorio **templates**, que contiene los ficheros fuente html: **base.html**, **cart.html**, **history.html**, **index.html**, **login.html**, **product.html**, **profile.html** y **register.html**.
(11) Directorio **thesessions**, subdirectorio de `app`, contiene los ficheros generados por las sesiones de Flask.
(12) Directorio **usuarios**, subdirectorio de `app`, que posee un directorio por cada usuario registrado en la aplicación (estas carpetas se van creando en el momento que se registra cada usuario), y dentro de cada uno, dos ficheros: **datos.dat** (donde se guardan los datos de registro del usuario) y **historial.json** (donde se guarda el historial de compra de cada usuario). En la imagen aportada se enseña un ejemplo de usuario registrado, *alumnodb*.
(13) & (14) Ficheros **__init__.py** y **__main__.py** que se encargan de establecer la configuración inicial de la aplicación (como puerto, IP, inicio de sesiones de Flask, etc.).
(15) Fichero **routes.py**, donde se encuentran definidos todos los *endpoints* que controlan la lógica del *backend*.

3 Ejecución en local

Para ejecutar la aplicación web en local se debe arrancar el *virtual environment* `si1pyenv` en primer lugar. Este entorno virtual ha sido creado bajo lo estipulado en la documentación de las prácticas. Se aporta en la raíz de la carpeta de la entrega un fichero **requirements.txt** con el cual podemos crear un *virtual environment* con los comandos `virtualenv si1pyenv` y `pip install -r requirements.txt`, así tendremos en el entorno virtual todo lo necesario para la ejecución.

Después de configurar y correr el entorno virtual, vamos al directorio donde se encuntra la carpeta **app** (en la carpeta de la entrega se encuentra en la raíz) y arrancamos la aplicación web con el comando `python -m app` o `python3 -m app`. Se adjunta una imagen a continuación con todo lo comentado en esta sección salvo por el hecho de que el entorno virtual ya se encontraba configurado previamente.



4 Ejecución bajo el servidor Apache

Para ejecutar la aplicación en el servidor Apache en los ordenadores del laboratorio se deberán seguir los siguientes pasos:
1. Descargar la entrega desde Moodle.
2. Asegurarse que dentro del directorio **app** están todas las carpetas especificadas en la sección "Estructura de carpetas y ficheros de la entrega", sobre todo las carpetas **usuarios** y **thesessions** que inicialmente se encuentran vacías y puede que no se preserven en GitHub.
3. Crear el entorno virtual **si1pyenv** en el directorio **public_html** de los ordenadores del laboratorio, con el comando `virtualenv si1pyenv --python=python3`.
4. Activar el entorno virtual con `source si1pyenv/bin/activate`.
5. Instalar todo lo necesario (importante no haberse saltado el paso 4), con el comando:
`pip3 install -r requirements.txt`.
6. Dar permisos a las carpetas donde Apache deberá realizar cambios. Para ello nos posicionaremos en el directorio **app** (`cd app`) con los comandos:
`setfacl -m u:www-data:rwx usuarios`
`setfacl -m u:www-data:rwx thesessions`
`setfacl -m u:www-data:rwx static/img`
7. Finalmente podremos acceder a la aplicación web escribiendo en el navegador:
`localhost/~<nia>/start.wsgi`

NOTA: Es posible que en algunos ordenadores del laboratorio el Apache este configurado para correr con Python2, por lo que puede fallar la aplicación al intentar correrse. Se ruega comprobar el *log* de Apache para indetificar este posible error.
En las futuras secciones se incluirán pantallazos del funcionamiento de la aplicación donde esta estaba corriendo en Apache (compruébese el enlace en el navegador).

5 Diseño del *Backend*

Para una eficiente programación, primero se ha de idear los *endpoints* necesarios para acoger toda la lógica interna de la aplicación web:

- **/index** o **/**: *endpoint* de la página principal o *landing page*. Si se accede con una petición `GET` simplemente muestra las últimas películas. En caso de realizar una búsqueda con algún filtro se accederá con una petición `POST`.

- **/register**: *endpoint* de la página de registro. Si se accede con una petición `GET` se mostrará el formulario (vacío) de registro. En cambio, si el usuario completa el formulario y supera las validaciones del *frontend* enviará una petición `POST` con los datos introducidos por el usuario.

- **/login**: *endpoint* de la página de inicio de sesión. De forma análoga a la página de registro, si se accede con una petición `GET` se mostrará el formulario de inicio de sesión y, en cambio, si el usuario ha introducido los datos de acceso, se realizará una petición `POST`.

- **/logout**: *endpoint* para cerrar sesión. No se muestra ninguna nueva página, simplemente se elimina la información de la sesión del usuario actual y se refresca la página principal.

- **/product/id**: *endpoint* de detalle de producto. Si se accede con una petición `GET` se mostrará el detalle de producto con el ID aportado. En cambio, si el usuario añade el producto al carrito se realizará una petición `POST` con la cantidad añadida.

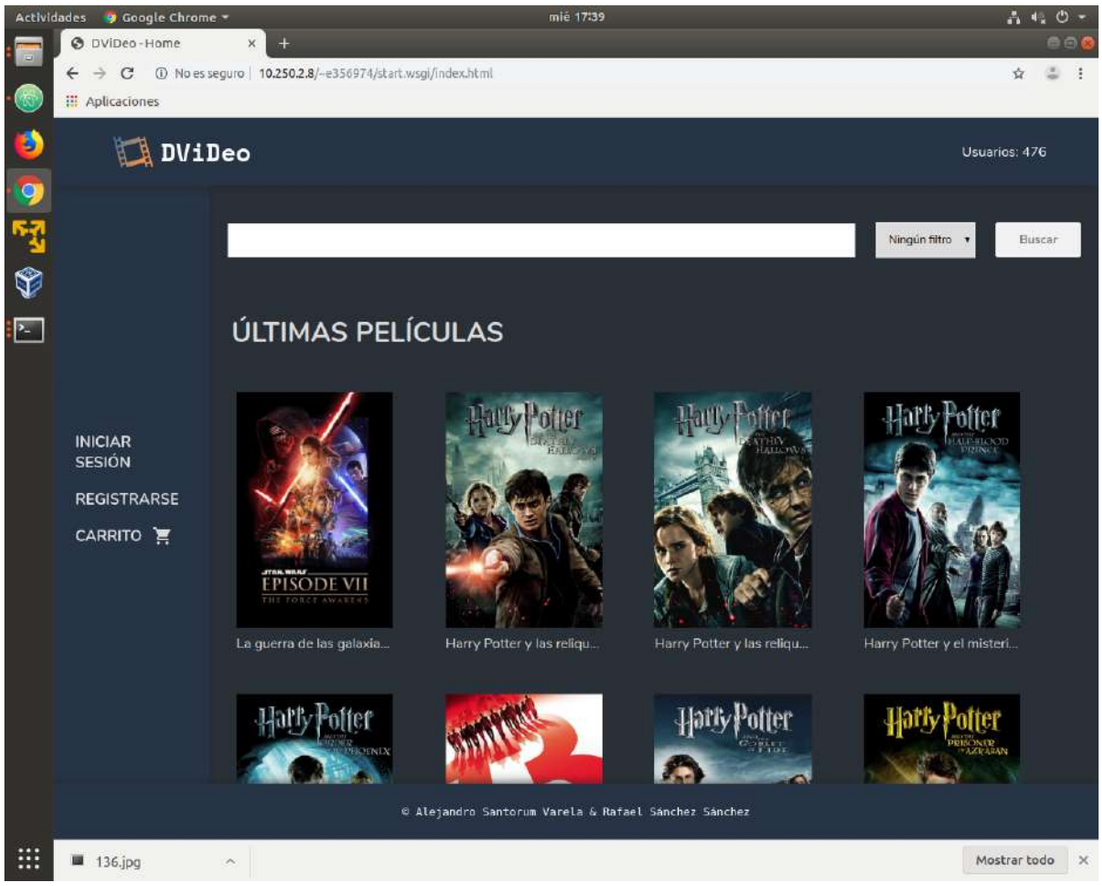
- **/cart**: *endpoint* para la página del carrito de la compra. Si se accede con una petición `GET` se mostrará los productos que actualmente se tienen añadidos a la cesta. Por otro lado, si el usuario elimina algún artículo de la cesta, se realizará una petición `POST` con el ID del producto que se desea eliminar.

- **/purchase**: *endpoint* para la lógica de compra. Sólo se pueden realizar peticiones `POST`. En la información de la sesión se encontrará todo el estado del carrito y de ahí se obtendrán los ID's y las cantidades de los productos que se desean comprar.
- **/history**: *endpoint* de la página del historial. Sólo se puede acceder con una petición `GET`. Se mostrará todo el historial de compra del usuario *logeado*, que se encuentra guardada en su correspondiente carpeta en el fichero `historial.json`.
- **/profile**: *endpoint* de la página de perfil del usuario que ha iniciado sesión. Accediendo con una petición `GET` se mostrará la página del perfil actual. Si el usuario desea cambiar su dirección de entrega o su saldo actual, se realizará una petición `POST`.
- **/connectedusers**: *endpoint* auxiliar para obtener el número aleatorio de usuarios que están usando la página en este momento. Este *endpoint* será usado por un script de `AJAX` para mostrar los usuarios conectados.

6 Funcionalidad de la aplicación web - DViDeo

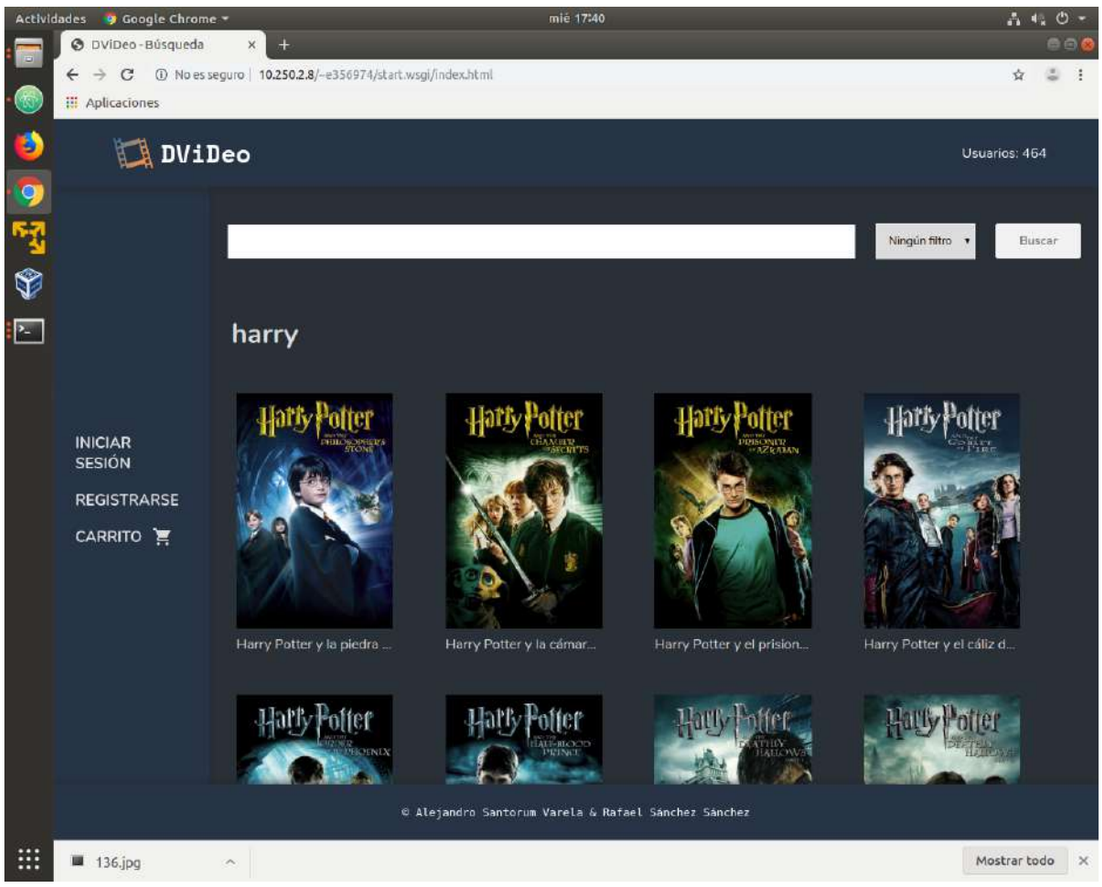
En esta sección mostraremos la funcionalidad de nuestra aplicación. Todas las imágenes adjuntas han sido tomadas con la aplicación corriendo bajo el servidor Apache.

Lo primero que nos encontramos nada más conectarnos es la página principal (`index.html`) sin usuario registrado:

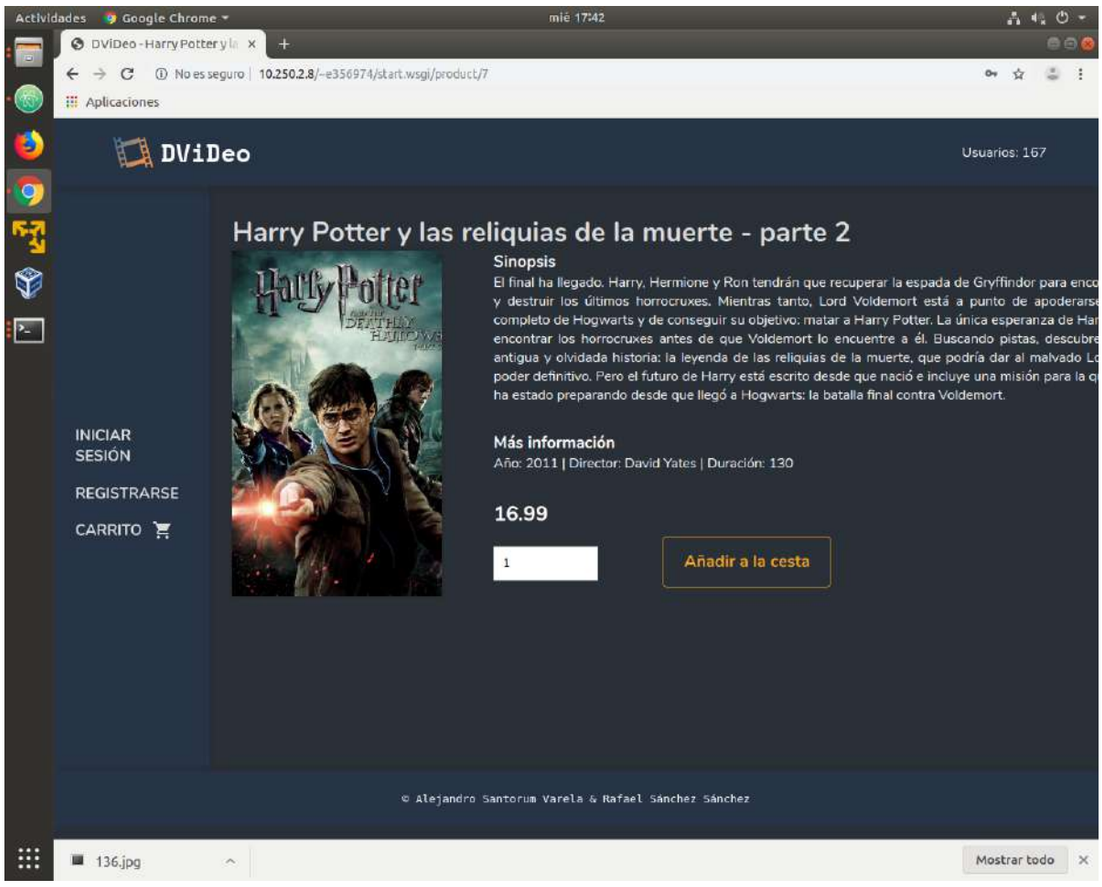


En la esquina superior izquierda se muestra el logo de la aplicación (`logo.svg`). En la esquina superior derecha se muestran los usuarios conectados (número aleatorio generado con el script `ajax_users.js`). En el menú lateral se incluyen los botones de Inicio de Sesión, Registro y Carrito (por si el usuario haya añadido algo sin siquiera *logearse*).

En la barra central de búsqueda podemos introducir una frase y se buscarán los títulos de películas que contengan dicha frase. También existe un filtro por categoría. A continuación se muestra un ejemplo de búsqueda con la frase 'harry'.

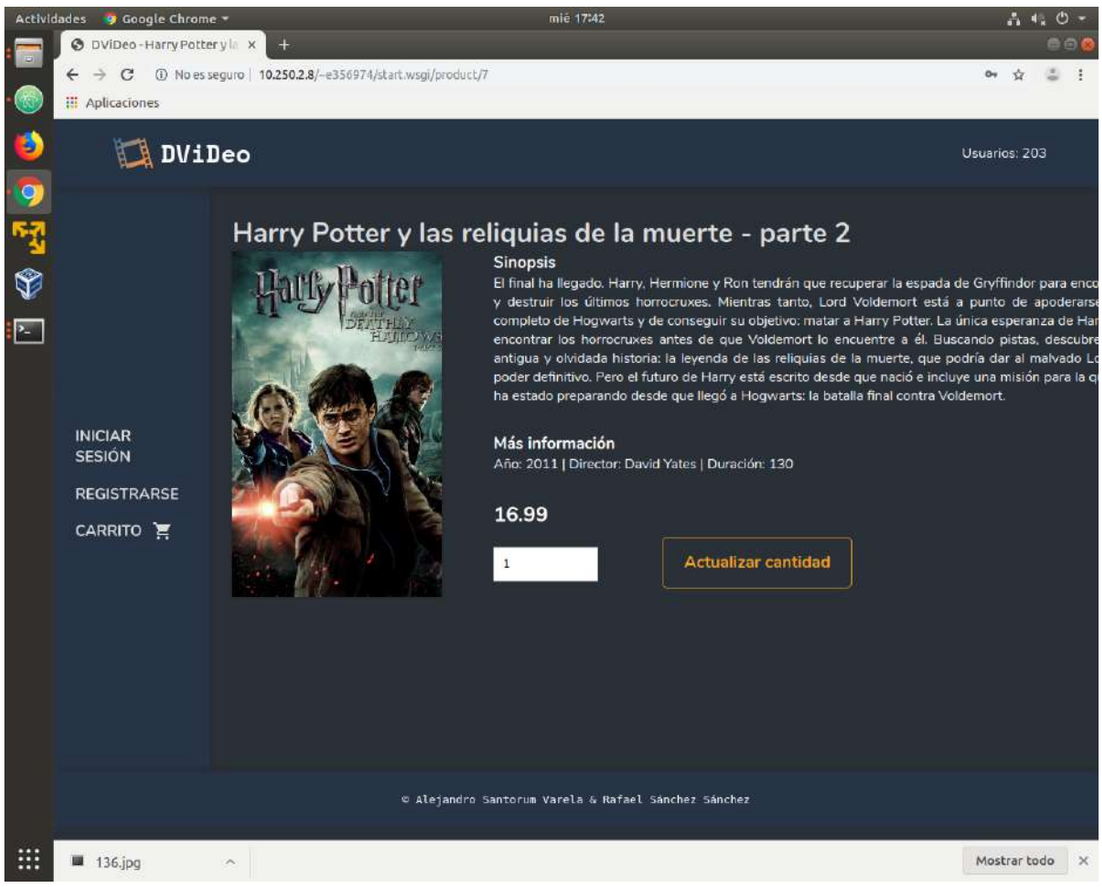


Si pinchamos encima del cartel de una película accedemos a su detalle (`product.html`):

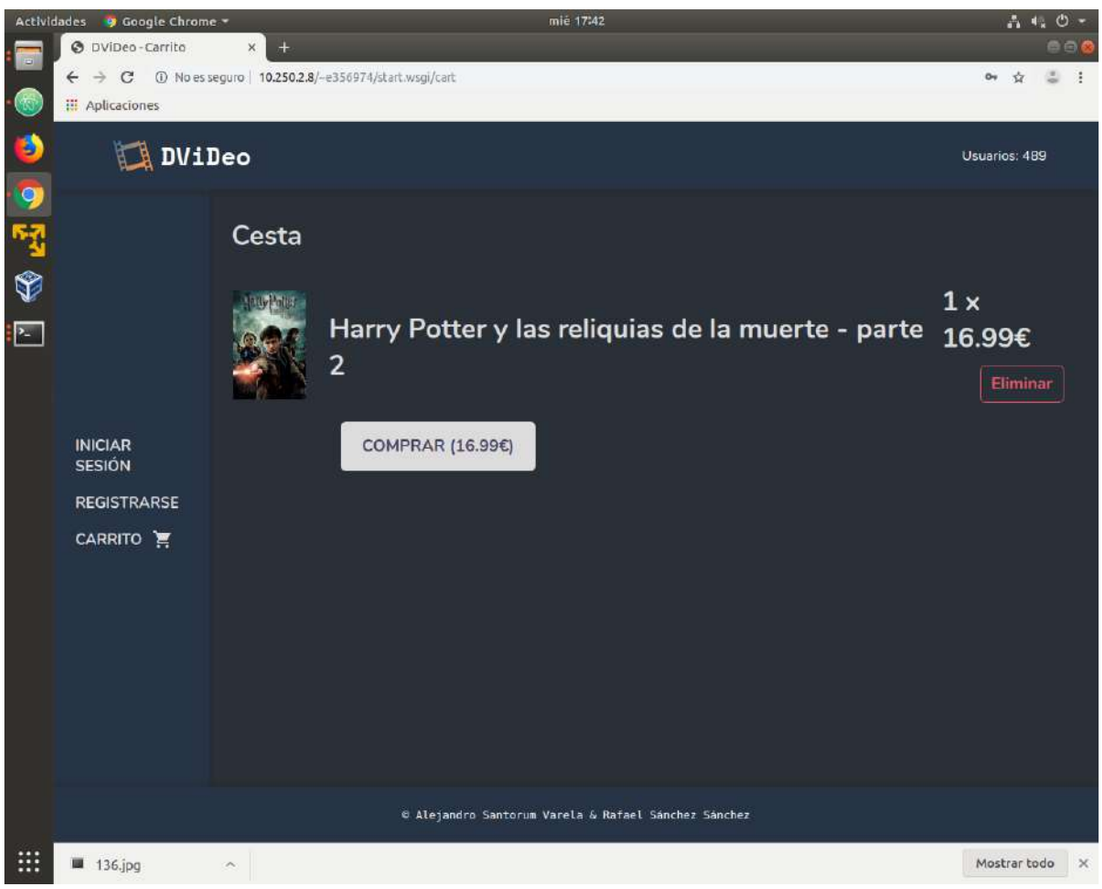


Aquí podemos ver la sinopsis de la película seleccionada, su año de estreno, director, etc. También su precio unitario y la posibilidad de añadir a la cesta más de una unidad.

Si tenemos la película ya en la cesta, se nos muestra la posibilidad de actualizar la cantidad añadida a la cesta:



Ahora veamos qué ocurre si nos vamos al carrito (`cart.html`):

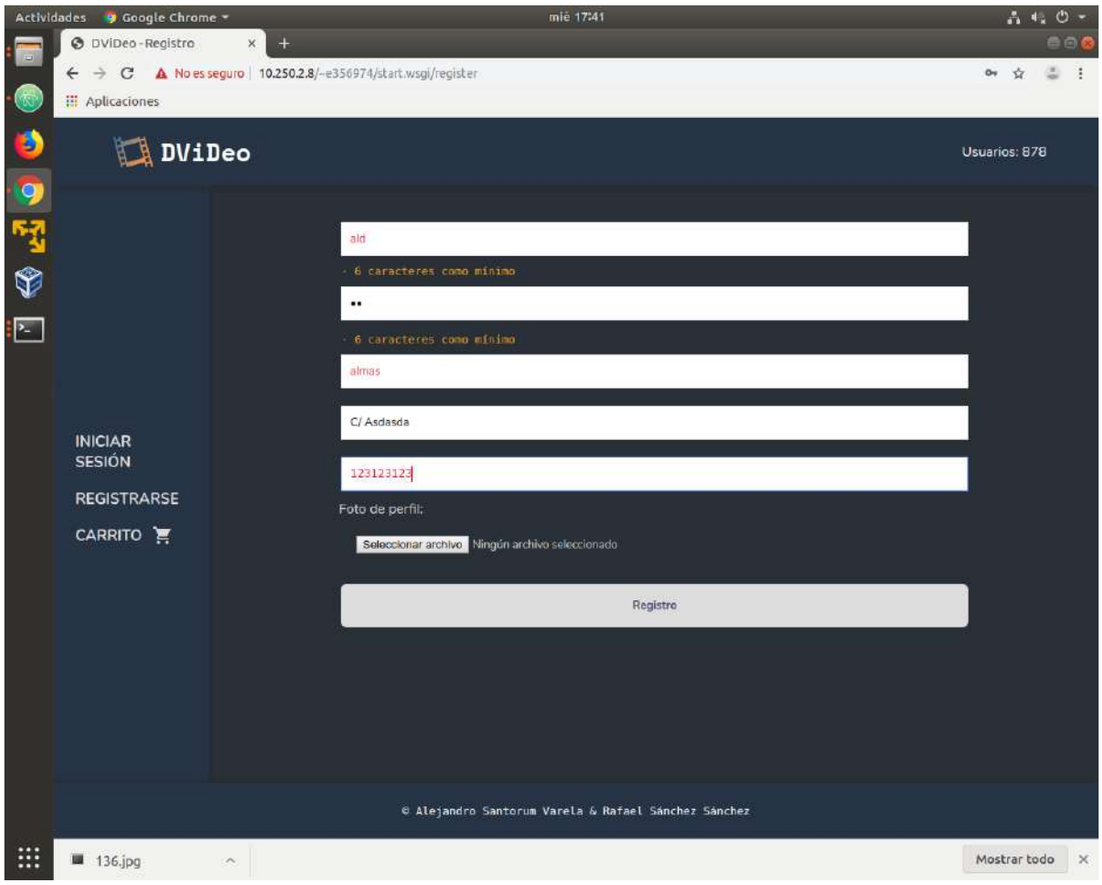


Se nos muestra la lista de películas añadidas a la cesta, la cantidad añadida por película, el precio unitario de cada una, un botón para eliminar la película añadida y finalmente un botón con el que podemos confirmar la compra, con el precio total de la misma. Si se presiona dicho botón cuando no se posee el saldo suficiente salta un mensaje de error que informa al usuario de eso.

Por otro lado, si el usuario aún no se ha registrado/iniciado sesión y pulsa el botón de

compra, se le redirigirá al formulario de inicio de sesión.

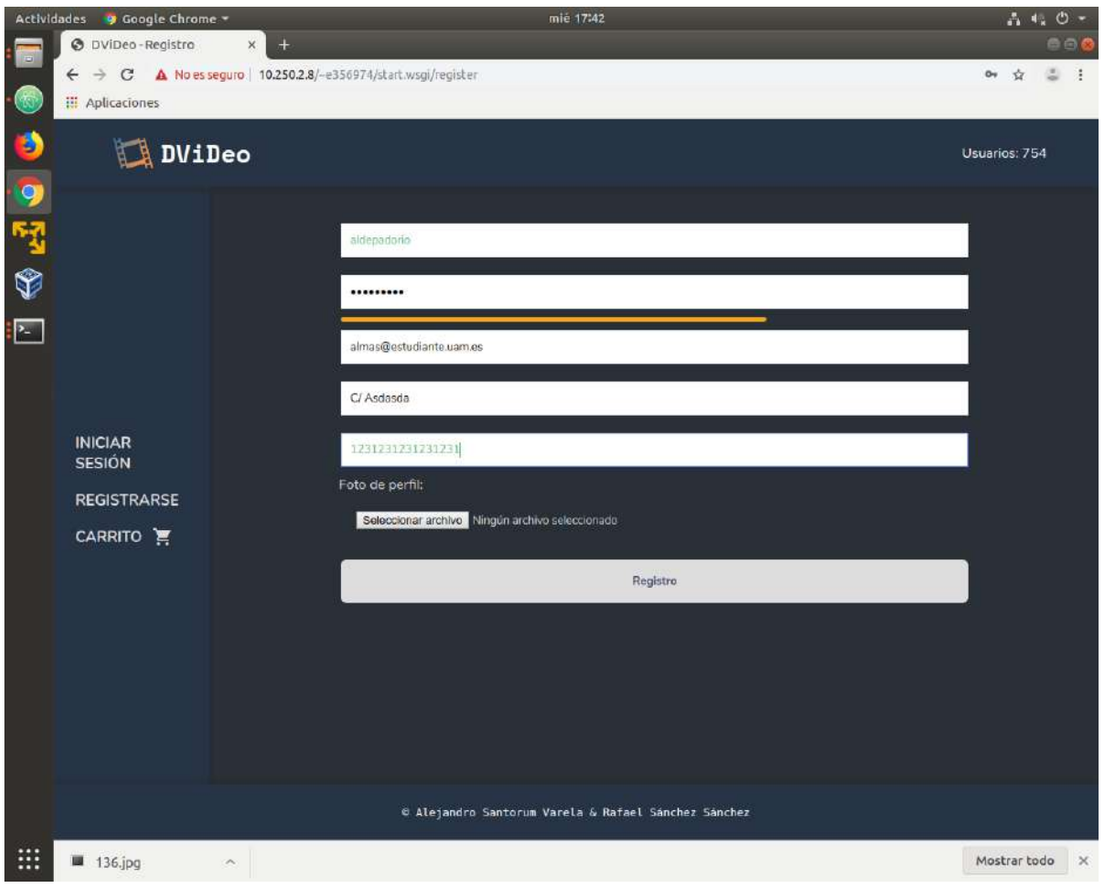
Veamos ahora el formulario para el registro de un usuario (`register.html`):



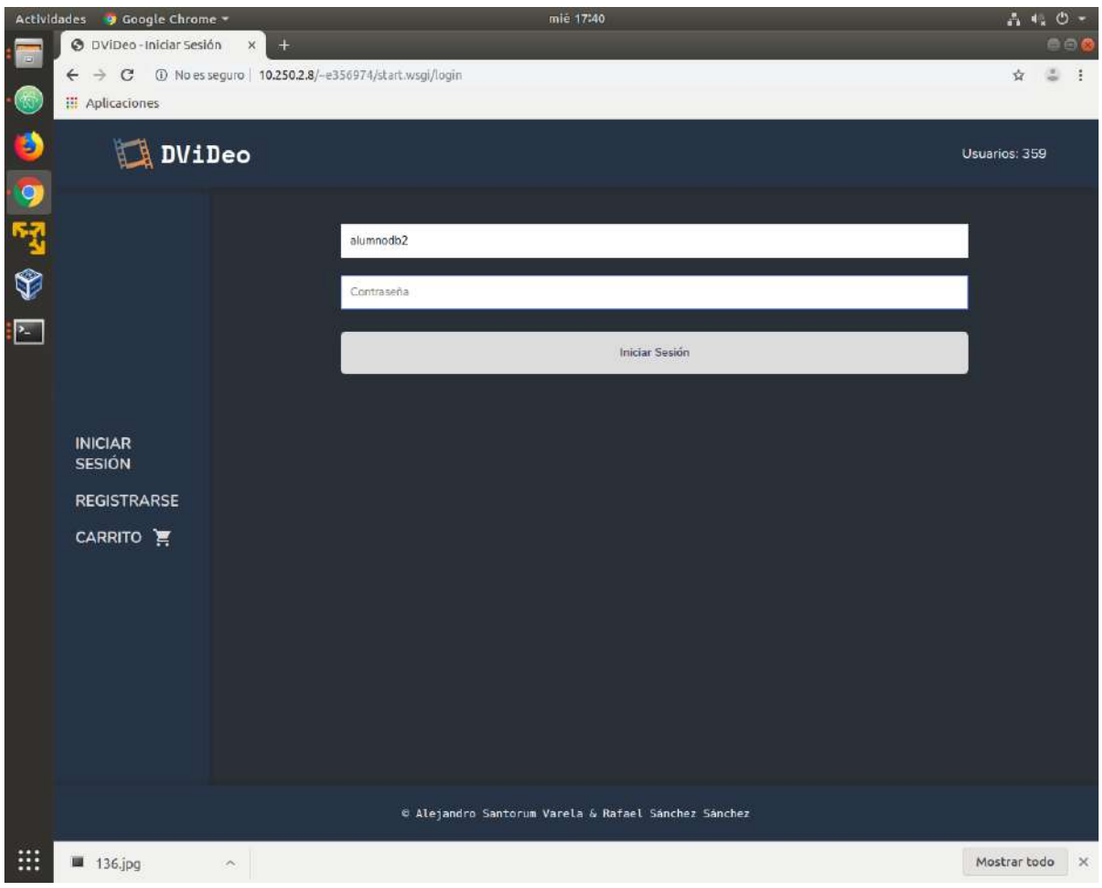
En la imagen anterior se muestra el formulario con datos incorrectos y la respuesta del *frontend* (implementado con funcionalidades de HTML5 y también con jQuery) ante dichos datos incorrectos de registro.

Para clarificar y ser más preciso, para poder registrarse se requiere un nombre de usuario con una **longitud mínima de 6 caracteres** y el **primer caracter tiene que ser uno alfanumérico**; la contraseña tiene que tener una **medidor de fortaleza** de la misma: baja fortaleza si solo tiene letras minúsculas (barra roja), fortaleza media si a parte de minúsculas tiene alguna mayúscula o algún número, y fortaleza fuerte si tiene tanto letras minúsculas, mayúsculas y números (todo esto comprobado con Query). Por otro lado, el correo electrónico debe tener un formato válido (comprobado con HTML5). Finalmente la tarjeta de crédito deben ser exactamente 16 números.

En el registro existe la posibilidad de añadir una foto de perfil, subiendo la foto desde el ordenador.

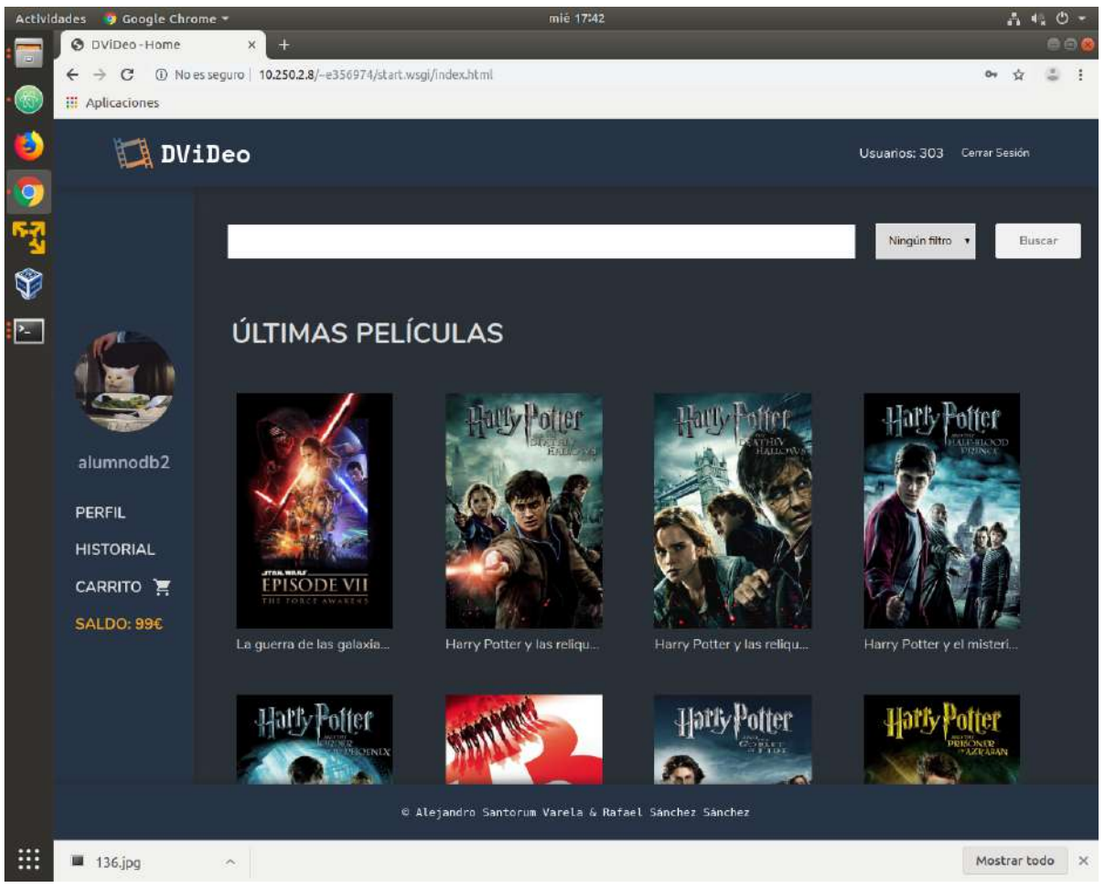


En la imagen anterior se muestra el formulario completado de forma adecuada. Presionando en "Registro" se nos abrirá la página principal de nuevo, donde podremos ir a la página de inicio de sesión (`login.html`):



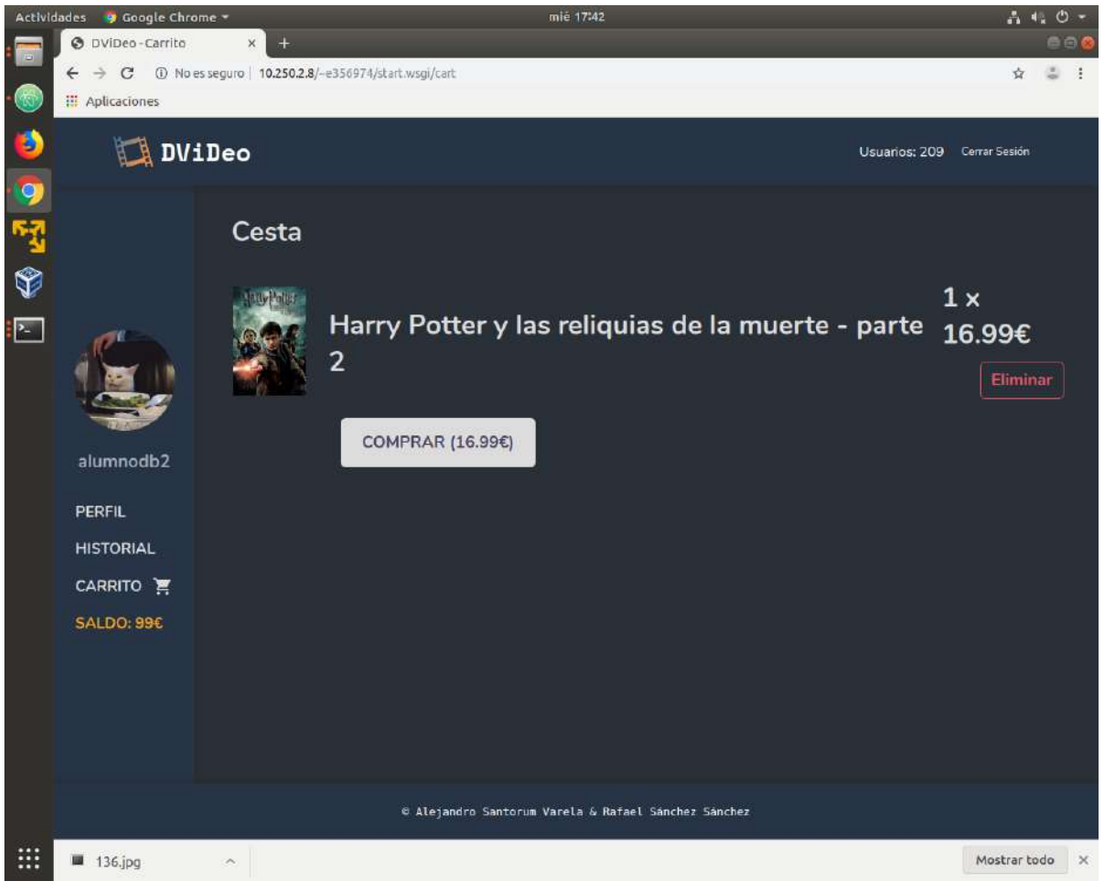
Aquí podremos introducir nuestros datos y *logearnos*. Por razones de simplicidad, la imagen anterior ha sido tomada después de iniciar sesión y cerrarla para mostrar el correcto funcionamiento de las *cookies*, ya que el campo del nombre de usuario ya se encuentra prerelleno.

Dicho esto, una vez *logeados* nos encontraremos con la página principal pero con un estado diferente:



En el menú lateral se incluye una foto de perfil (por defecto se aporta una, en caso de que en el registro no se subiera ninguna), enlaces para dirigirnos al perfil, historial, carrito y el saldo actual. Por otro lado, en el menú superior se siguen mostrando los usuarios conectados y el botón para cerrar sesión.

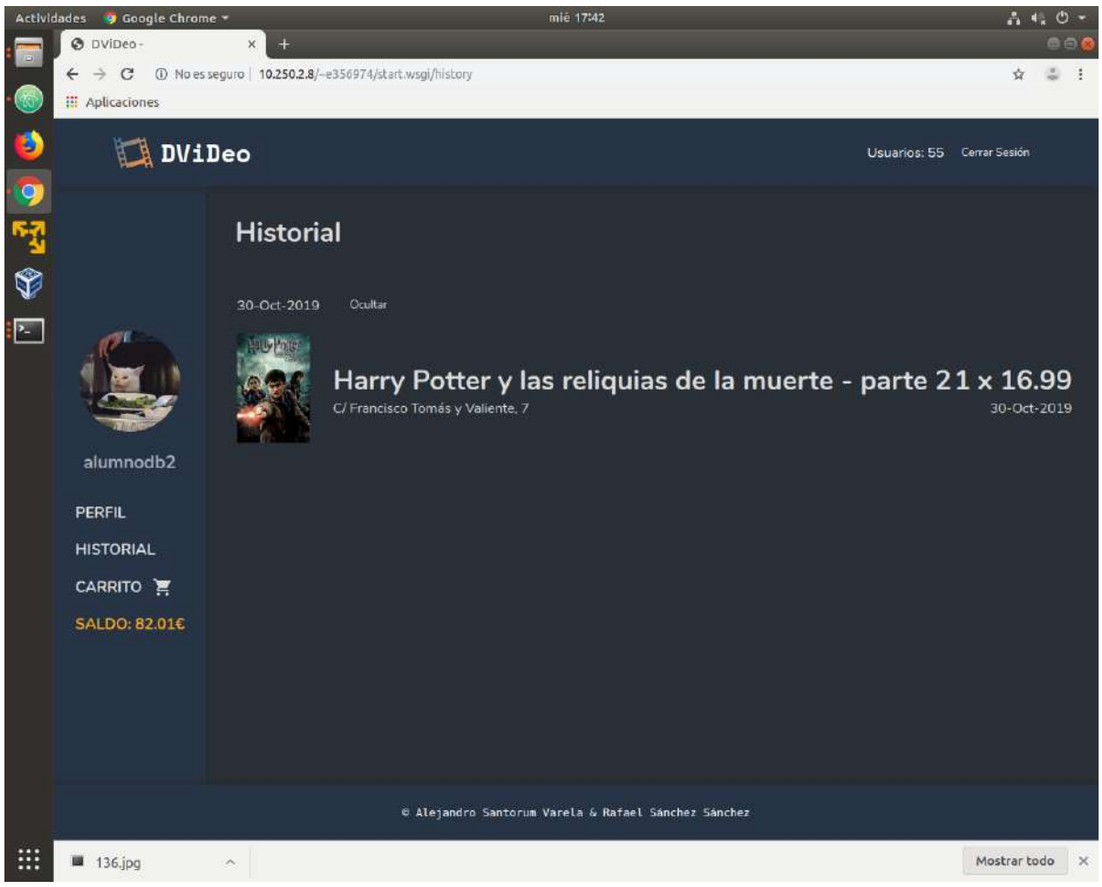
En este momento ya estamos listos para realizar la compra de los productos que habíamos añadido:



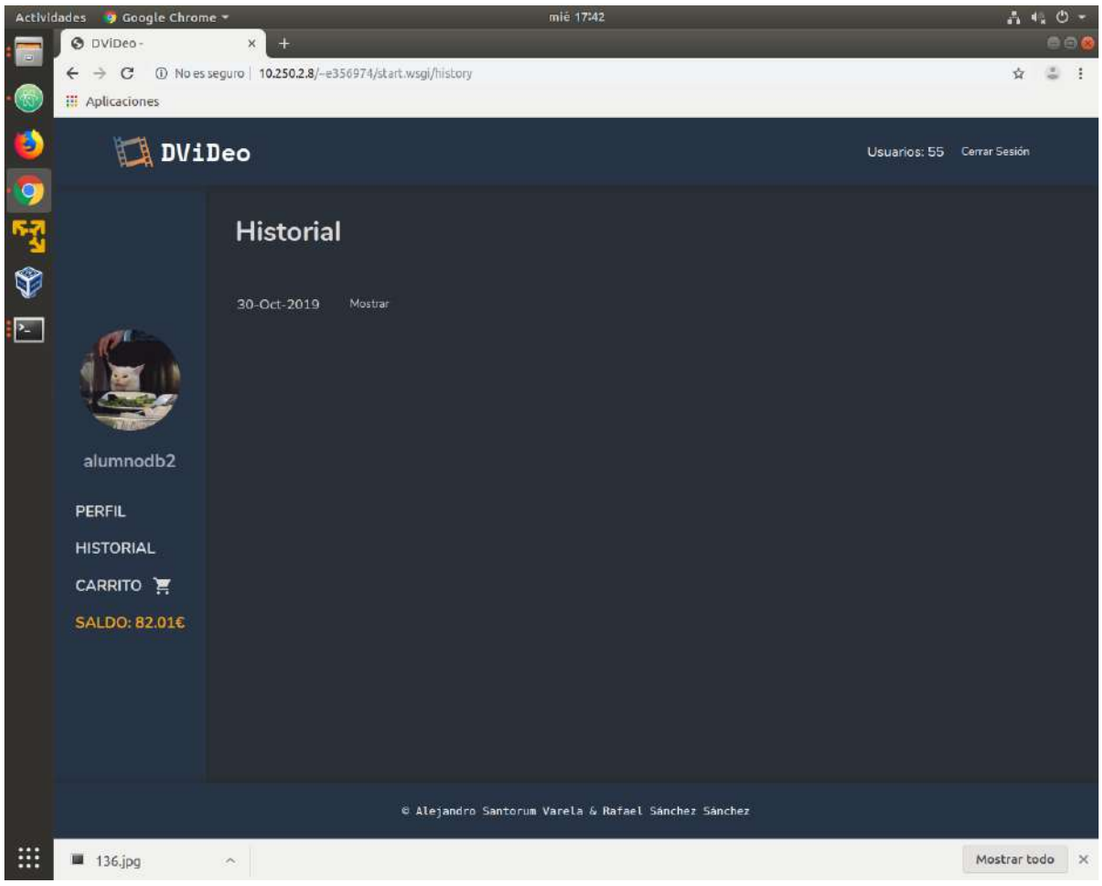
En la imagen anterior se pueden ver los productos que habían sido añadidos antes incluso de iniciar sesión, viendo que el funcionamiento de las sesiones de Flask y el guardado de datos

en los ficheros del usuario ha sido el esperado.

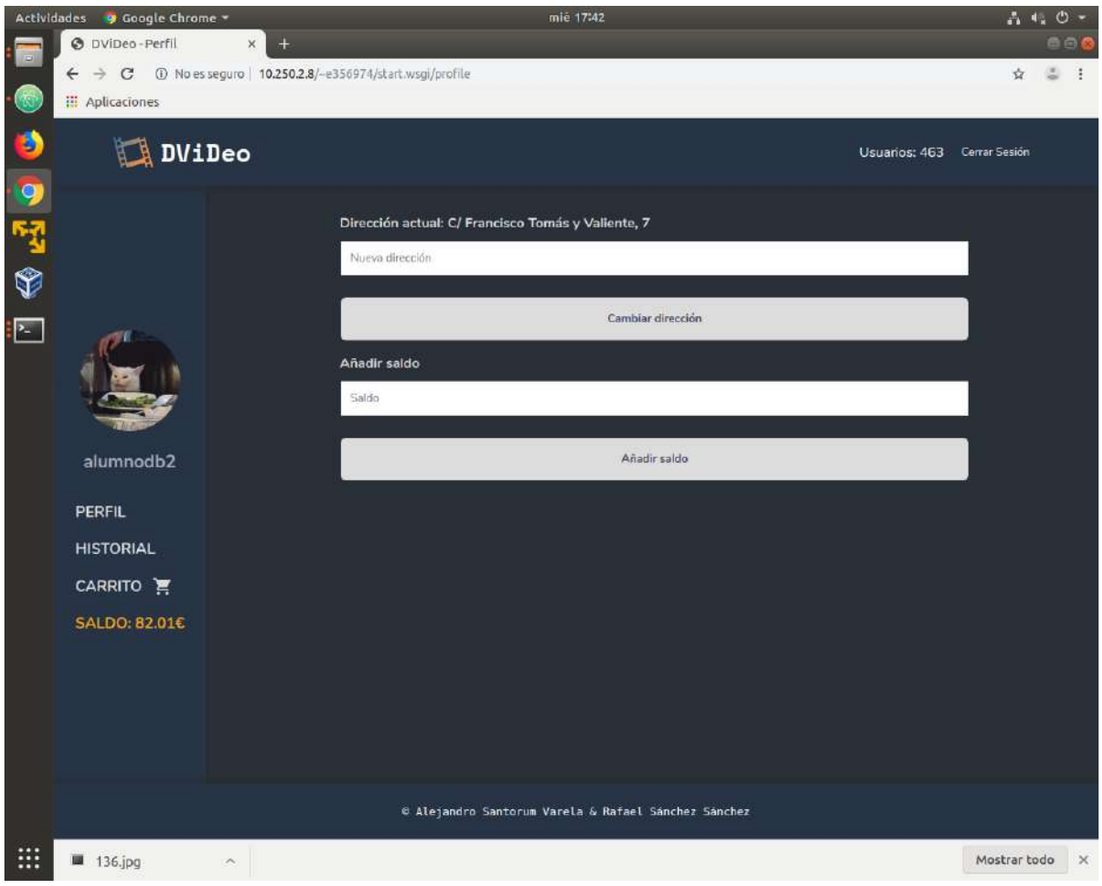
Finalizamos la compra pulsando en "Comprar" y si tenemos el saldo suficiente se nos redirigirá a la página principal con el saldo e historial ya actualizado. Esto último lo podemos comprobar pinchando en "Historial" en el menú lateral (`history.html`):



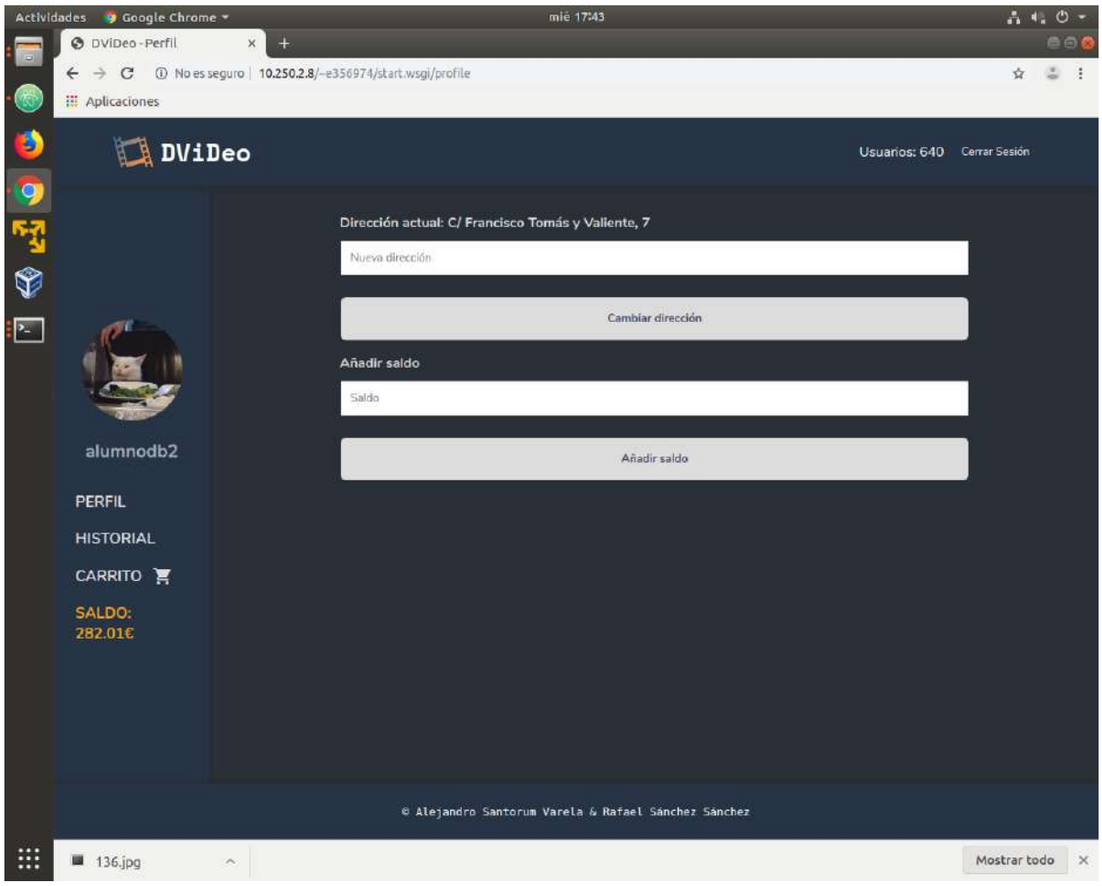
Se puede comprobar que en el historial se incluye la compra realizada recientemente, con la fecha, cantidad, precio unitario y dirección de entrega. Además, se aporta un botón por el cual podremos ocultar la lista de productos comprados en una misma fecha:



Finalmente, podremos ir a nuestro perfil pulsando en "Perfil" (`profile.html`):



Donde podremos cambiar la dirección de envío y aumentar nuestro saldo actual:



Hemos incrementado el saldo en 200.

Comentar finalmente que tanto el menú superior, el menú lateral y el *footer* se han incluido en el fichero `base.html` y de ahí se exportan al resto de ficheros HTML.

7 Conclusiones

Llegado el final de esta práctica es el momento de recapitular. En la primera práctica se ha desarrollado el *frontend* y en esta principalmente el *backend* utilizando el *framework* de Python `Flask`.

Ahora nos centramos ya en la siguiente, donde integraremos todos los datos de usuarios y películas en una base de datos relacional.

8 Bibliografía y lugares de referencia de código

- Tutorial Flask `https://www.tutorialspoint.com/flask/index.htm`
- Flask cookies `https://www.tutorialspoint.com/flask/flask_cookies.htm`
- Flask sessions `https://pythonhosted.org/Flask-Session/`
- Tutorial Javascript `https://www.w3schools.com/js/`
- WSGI con Flask `http://flask.pocoo.org/docs/0.12/deploying/mod_wsgi/`
- Tutorial CSS `https://internetingishard.com/html-and-css/`
- Validacion con jQuery `https://formden.com/blog/validate-contact-form-jquery`
- Consultas de código en general `https://stackoverflow.com/`