MEMORIA PRÁCTICA 3

Alejandro Santorum Varela - alejandro.santorum@estudiante.uam.es
Rafael Sánchez Sánchez - rafael.sanchezs@estudiante.uam.es
Sistemas Informáticos I
Práctica 3 Pareja 9

27 de noviembre de 2019

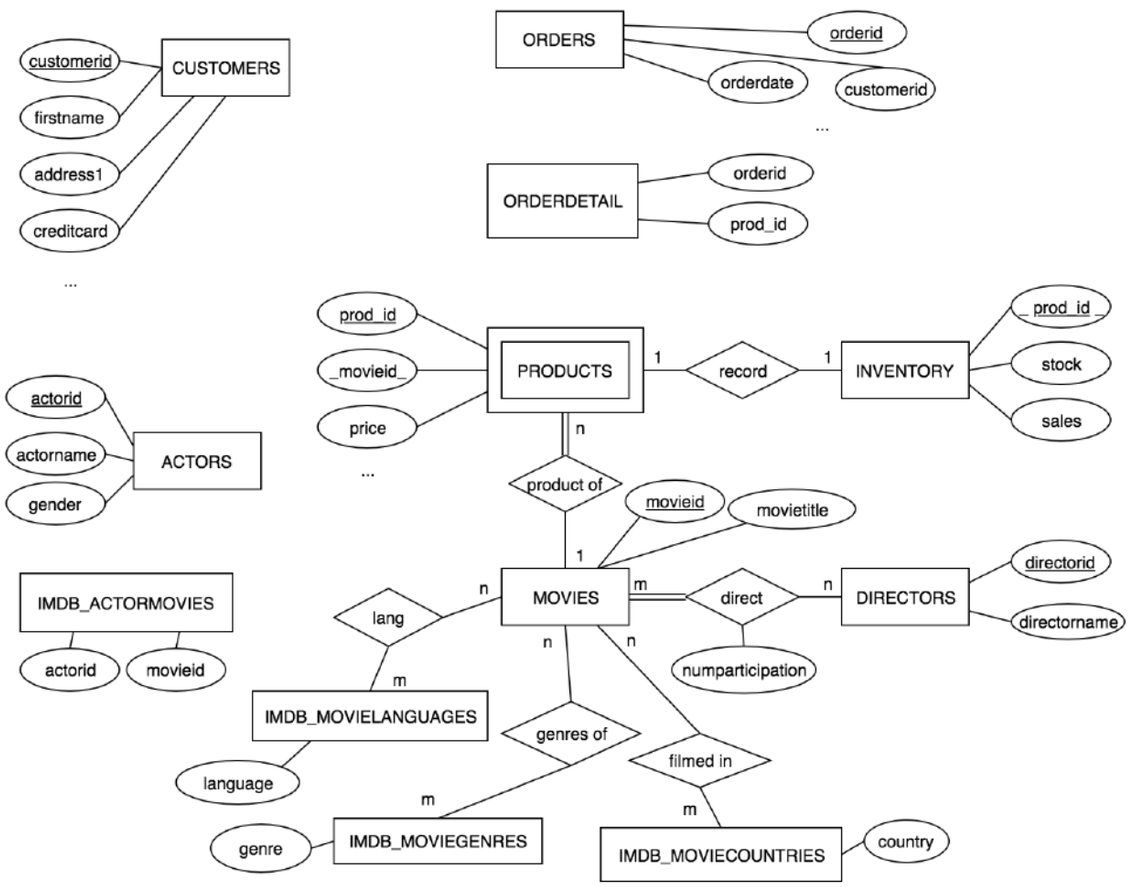
Contents

1 Introducción

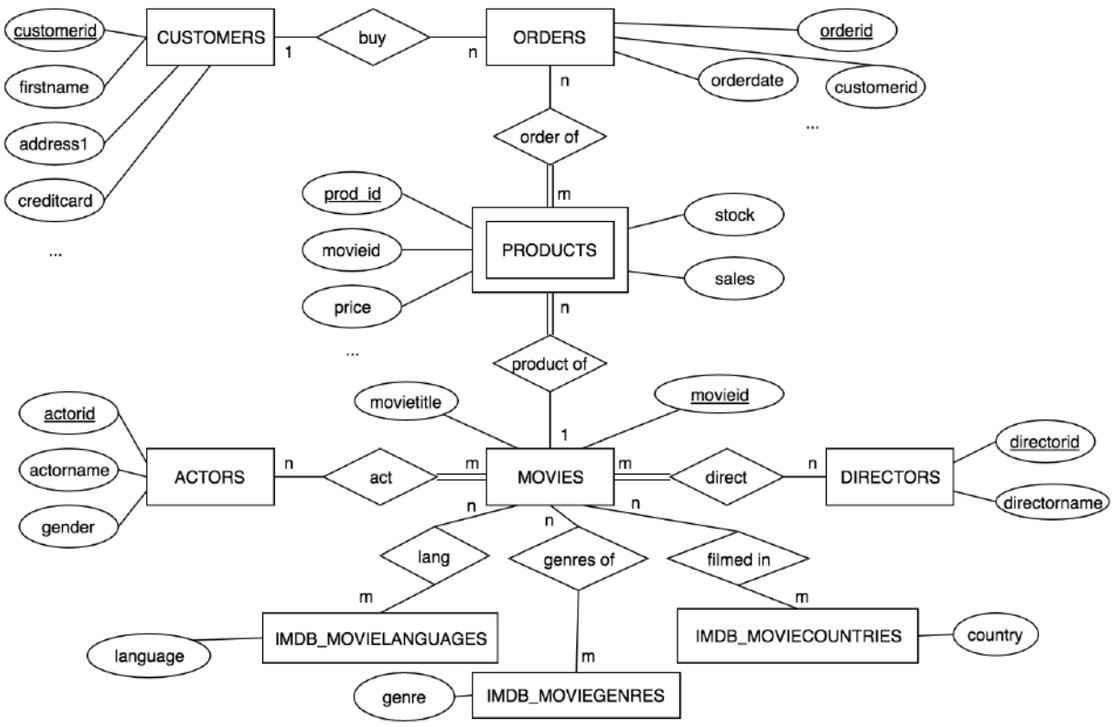
Nos encontramos ahora en la tercera práctica de Sistemas Informáticos I. En esta nos centraremos en la implementación de la base de datos (en PostgreSQL) que dará apoyo al *backend* y donde se guardarán los datos de los productos, usuarios y compras de nuestra página web.

2 Diagramas ER

A continuación mostramos el diagrama ER de la base de datos aportada en la práctica. Es importante observar que este diagrama se ha hecho realizando ingeniería inversa sobre la base de datos inicial y que, aunque exista un campo orderid en orderdetail y otro en orders (por ejemplo), no se a considerado que estos campos instancian una relación ya que en la tabla orderdetail orderid NO es una *foreign key*. Este ejemplo se puede extrapolar al resto de *primary keys* y *foreign keys* del resto de tablas.



Y ahora, mostramos el aspecto del diagrama Entidad-Realción después de ejecutar el fichero SQL **actualiza.sql**:



Como se puede ver, se han modificado varias relaciones, debido a la corrección de varias *foreign keys* o *primary keys*. El resto de cambios de actualiza.sql no son visibles desde el diagrama, pero sí desde la propia base de datos y su funcionamiento.

3 Estructura de carpetas y ficheros de la entrega

En el directorio raíz podremos encontrar los siguientes directorios:

- directorio **app** (1)
- directorio **srcsql** (2)
- fichero **requirements.txt** (3)
- directorio **diagrams** (4)
- fichero **Memoria-P3.pdf** (5)

La utilidad del fichero **requirements.txt** (3) se comenta en la sesión de ejecución en local.
El fichero **Memoria-P3.pdf** (5) es esta propia memoria.
El directorio **diagrams** (4) contiene los dos diagramas ER solicitados e incluidos en esta memoria.
El directorio **srcsql** (2) contienen todos los ficheros fuente SQL: `actualiza.sql`, `getTopMonths.sql`, `getTopVentas.sql`, `setOrderAmount.sql`, `setPrice.sql`, `updOrders.sql`, `updInventory.sql` y `exec_all_files.sql` (este último no se solicitaba, pero se entrega para poder ejecutar todos los ficheros SQL de este mismo directorio a la vez y en el **orden correcto**).
Finalmente, el directorio **app** (1) contiene el resto de directorios y ficheros fuente de la aplicación web:

- fichero **__main__.py** (6)
- fichero **__init__.py** (7)
- fichero **routes.py** (8)
- fichero **database.py** (9)
- directorio **templates** (10)

- directorio **static** (11)
- directorio **thesessions** (12)

Los ficheros __main__.py (6) y __init__.py (7) se encargan del lanzamiento de la aplicación.
Los ficheros fuente `routes.py` (8) y `database.py` (9) se encargan, respectivamente, de instanciar la lógica de los *endpoints* de la aplicación y de realizar la conexión y comunicación con la base de datos textbfsi1 de PostgreSQL.
En el directorio `templates` (10) se encuentran los ficheros fuente HTML.
En el directorio `static` (11) se encutran los ficheros fuente de CSS, Javascript y los posibles imágenes que necesitemos.
Finalmente, el directorio `thesessions` (12) incluye los posibles datos de las sesiones de Flask que utiliza nuestra aplcación.

4 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`, arrancar con `source si1pyenv/bin/activate` e instalar todo lo necesario para ejecutar la práctica con `pip install -r requirements.txt`, así tendremos en el entorno virtual todo lo necesario.

Después de configurar y correr el entorno virtual, deberemos crear la base de datos que vamos a utilizar con el comando aportado en la documentación de la práctica: `createdb -U alumnodb si1` seguido de `gunzip -c dump_v1.3.sql.gz | psql -U alumnodb si1` para llevarla con los datos iniciales. (Importante: se necesita tener descargado el fichero dump_v1.3.sql.gz).

Una vez creada la base de datos, vamos al directorio **srcsql** que se encuentra en la raíz de la carpeta de entrega, donde se encuetran los ficheros fuente SQL solicitados. Una vez aquí accederemos al intérprete de PostgreSQL (`psql si1`) para ejecutar los ejercicios SQL desarrollados. Podemos ejecutarlos por el orden en que aparecen en la documentación de la práctica (i.e. \i actualiza.sql; \i setPrice.sql; etc.) o **ejecutando directamente el script SQL aportado** para ejecutar todos los ejercicios directamente: \i exec_all_files.sql;.

Observación: Algunas consultas/actualizaciones sobre la base de datos requieren su tiempo de ejecución. En nuestras máquinas NO tarda TODO más de 20 segundos, pero se ruega tener calma en un posible ordenador no muy potente.

Una vez ejecutados los ejercicios SQL, podremos salir del intérprete de PostgreSQL y volviendo a la raíz de la carpeta de la entrega (donde se encuentran las carpetas srcsql o app) podremos ejecutar la aplicación web con el comando **python3 -m app**.

A partir de aquí la aplicación está completamente operativa en **127.0.0.1:5000**.

5 Destacadas decisiones de diseño y observaciones

A continuación comentamos las decisiones de diseño que merecen ser más resaltadas:

- El email de los usuarios es único y no vacío.
- Importante leer el fichero actualiza.sql para entender los cambios realizados en la base de datos entregada.
- Existian varios pares (orderid, prod_id) iguales pero con diferente cantidad en orderdetail, por lo que se han sumado dichas cantidades para tener un único par (orderid, prod_id) que es una *primary key* en la tabla orderdetail.

- Se ha juntado la tabla Inventory con la tabla Products debido a que era una tabla redundante y toda su información se podría incluir en dos columnas en la tabla Porducts.
- Se han mejorado las relaciones IMDB_MOVIEGENRES, IMDB_MOVIELANGUAGES y IMDB_MOVIECOUNTRIES tal y como se dice en el enunciado para que la base de datos pese menos.
- Se ha añadido una columna de saldo o *cash* en la tabla Customers para guardar el saldo actual de cada usuario.
- Se ha creado la tabla Alertas tal y como se pide en el enunciado, pero su uso es opaco para la aplicación web (al menos para esta práctica).
- Se crea/crean *view/s* para que las posteriores *queries* sean más rápidas y más claras.
- Se ha modificado el tamaño máximo de region en la tabla Customers de 6 caracteres como máximo a 32.
- Las secuencias de orderid y customerid se han configurado para empezar a partir del correspondiente ID más alto de la base de dato con su estado actual.
- Se han configurado el *netamount* y el *totalamount* de la tabla Orders con el valor por defecto 0'0.
- Se ha confugurado el campo *tax* de la tabla Orders con el valor por defecto 15.
- Importante, si se ejecutan los ejercicios SQL uno a uno, hay que ejecutarlos por el orden especificado en el enunciado y, entre el ejercicio setOrderAmount.sql y getTopVentas.sql HAY QUE EJECUTAR `select setOrderAmount();`
- No se utilizan transacciones a la hora de realizar la compra porque no correspondía para esta práctica, pero se ha tenido en cuenta. Es importante recordar que existe un hipotético caso en el que dos usuarios quieran comprar el mismo producto a la vez y exista suficiente *stock* para uno de ellos pero esta comprobación se realice simultáneamente y, por lo tanto, ambas compras se efectuarían debido a una desafortunada concurrencia y la stock final sería negativa.

6 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:

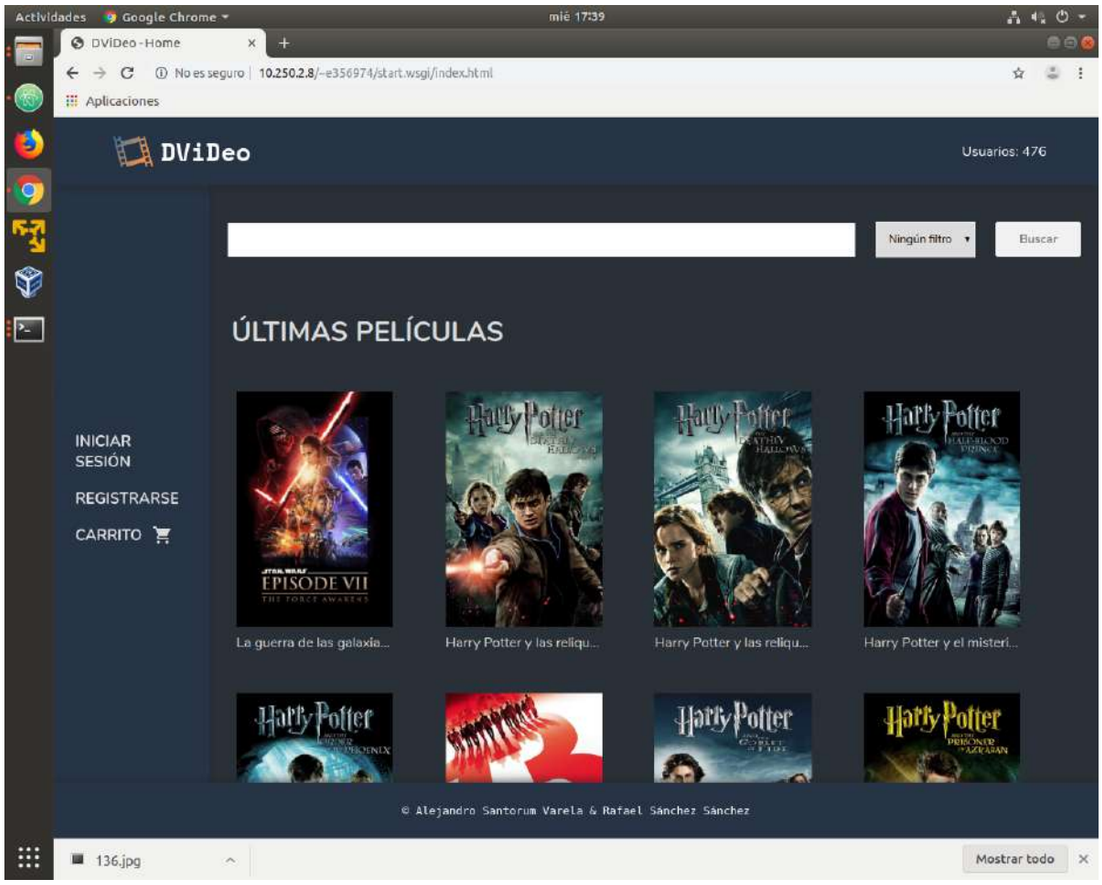
- **/index** o **/**: *endpoint* de la página principal o *landing page*. Si se accede con una petición `GET` simplemente muestra las últimas películas. En caso de realizar una búsqueda con algún filtro se accederá con una petición `POST`.
- **/register**: *endpoint* de la página de registro. Si se accede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

7 Funcionalidad de la aplicación web - DViDeo

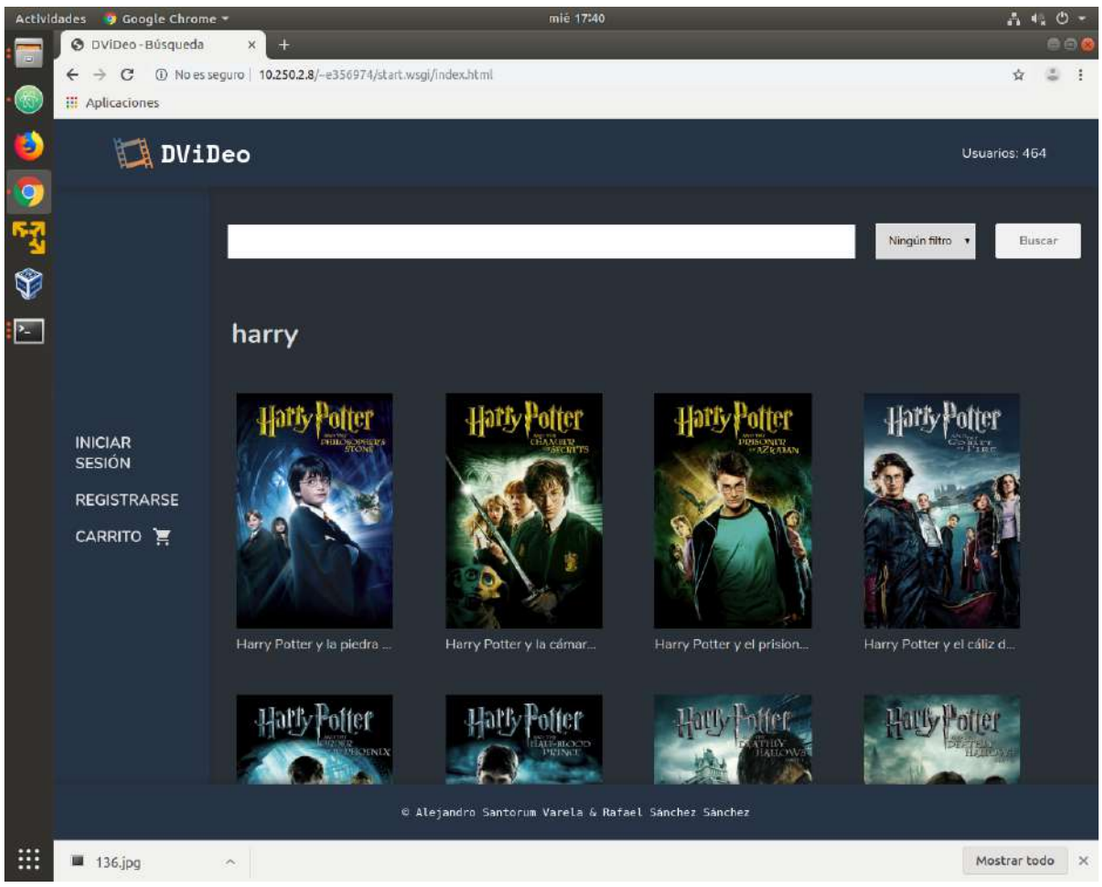
En esta sección mostraremos la funcionalidad de nuestra aplicación. Todas las imágenes adjuntas han sido tomadas con la aplicación corriendo bajo el servidor Apache.

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

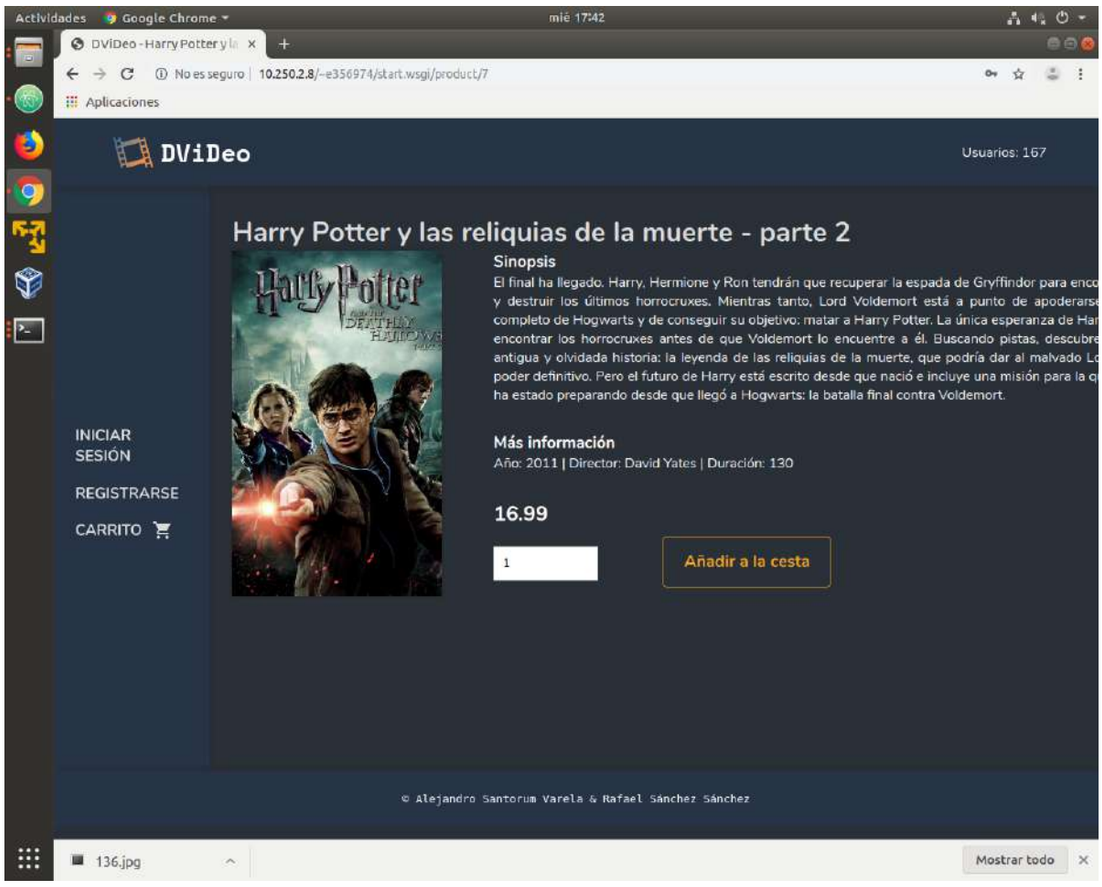


En la esquina superior izquierda se muestra el logo de la aplicación (`logo.svg`). En la esquina superior derecha se muestran los usuarios conectados (número aleatorio generado con el script `ajax_users.js`). En el menú lateral se incluyen los botones de Inicio de Sesión, Registro y Carrito (por si el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

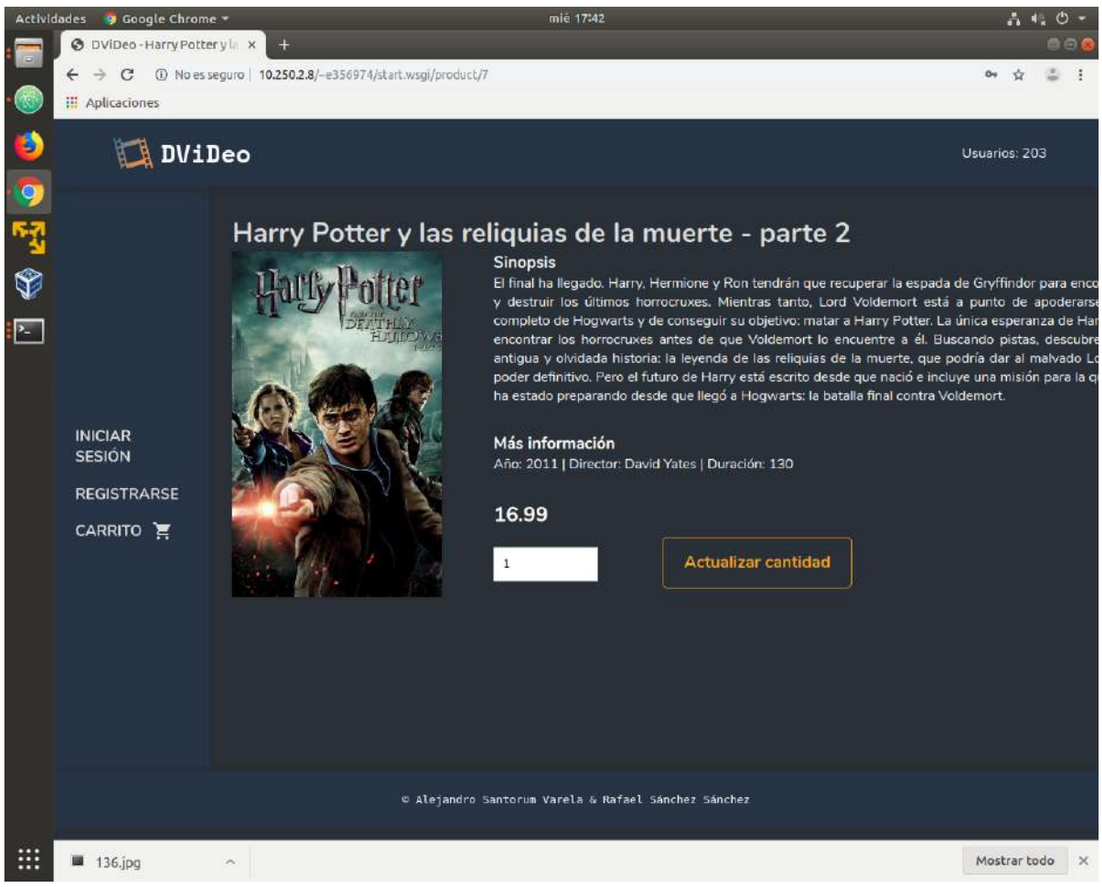


Si pinchamos encima del cartel de una película accedemos a su detalle (`product.html`):

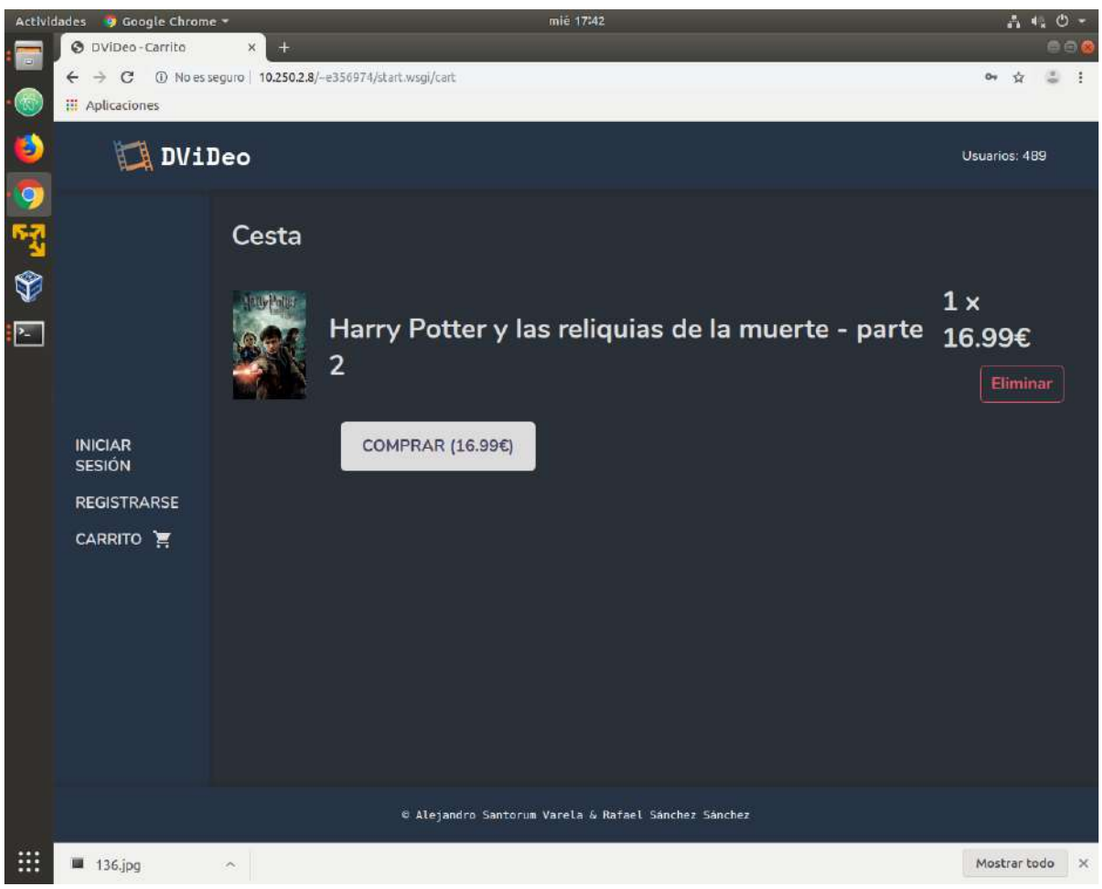


Aquí podemos ver la sinopsis de la película seleccionada, su año de estreno, director, etc. También su precio unitario y la posibilidad de añadir a la cesta más de una unidad.

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`):

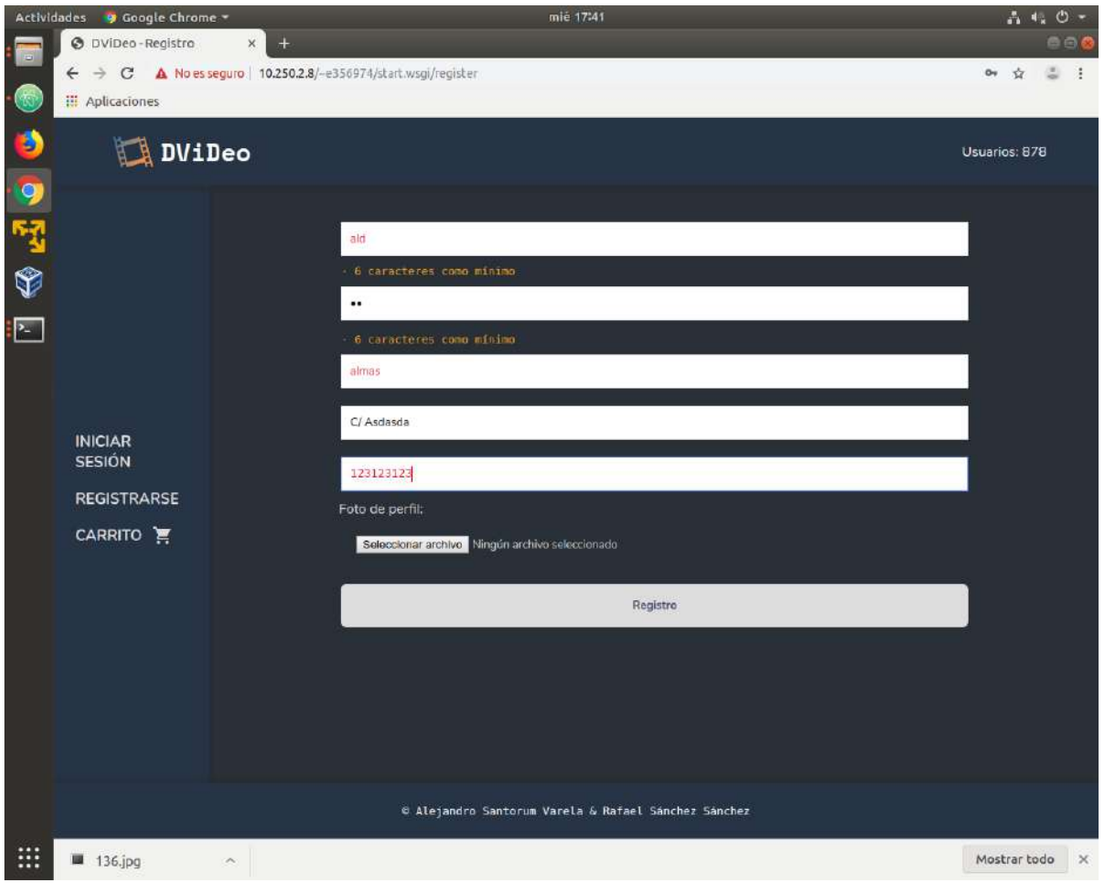


Se nos muestra la lista de películas añadidas a la cesta, la cantidad añadida por película, el precio unitario de cada una, un botón para eliminar la película añadida y finalmente un botón con el que podemos confirmar la compra, con el precio total de la misma. Si se presiona dicho botón cuando no se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

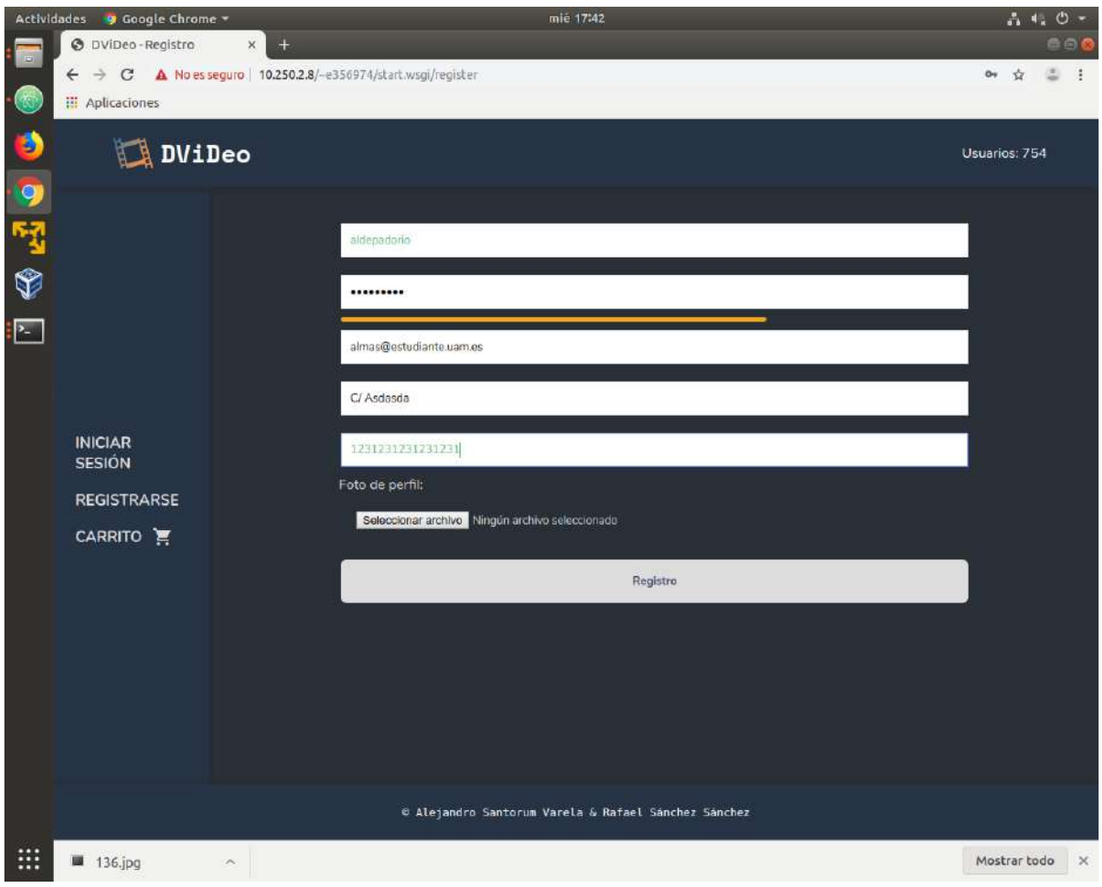
Veamos ahora el formulario para el registro de un usuario (`register.html`):



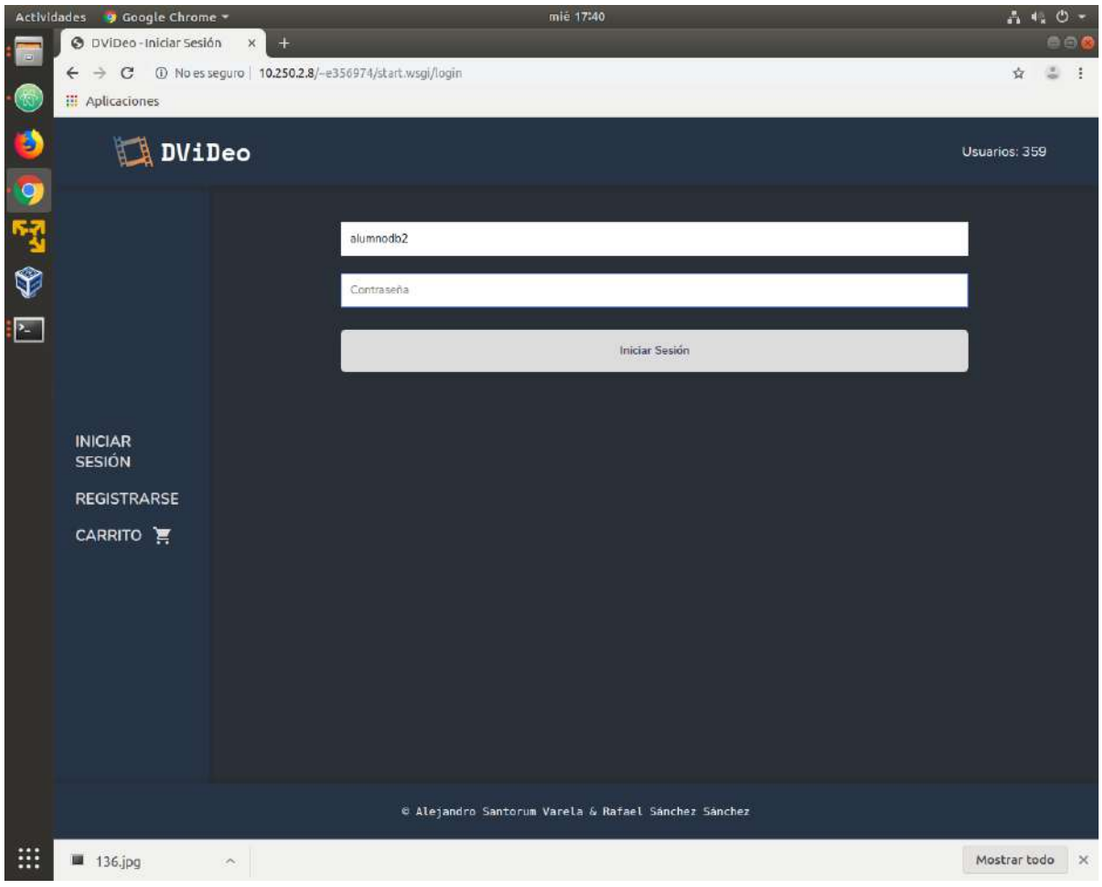
En la imagen anterior se muestra el formulario con datos incorrectos y la respuesta del *frontend* (implementado con funcionalidades de HTML5 y también con jQuery) ante dichos datos incorrectos de registro.

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.

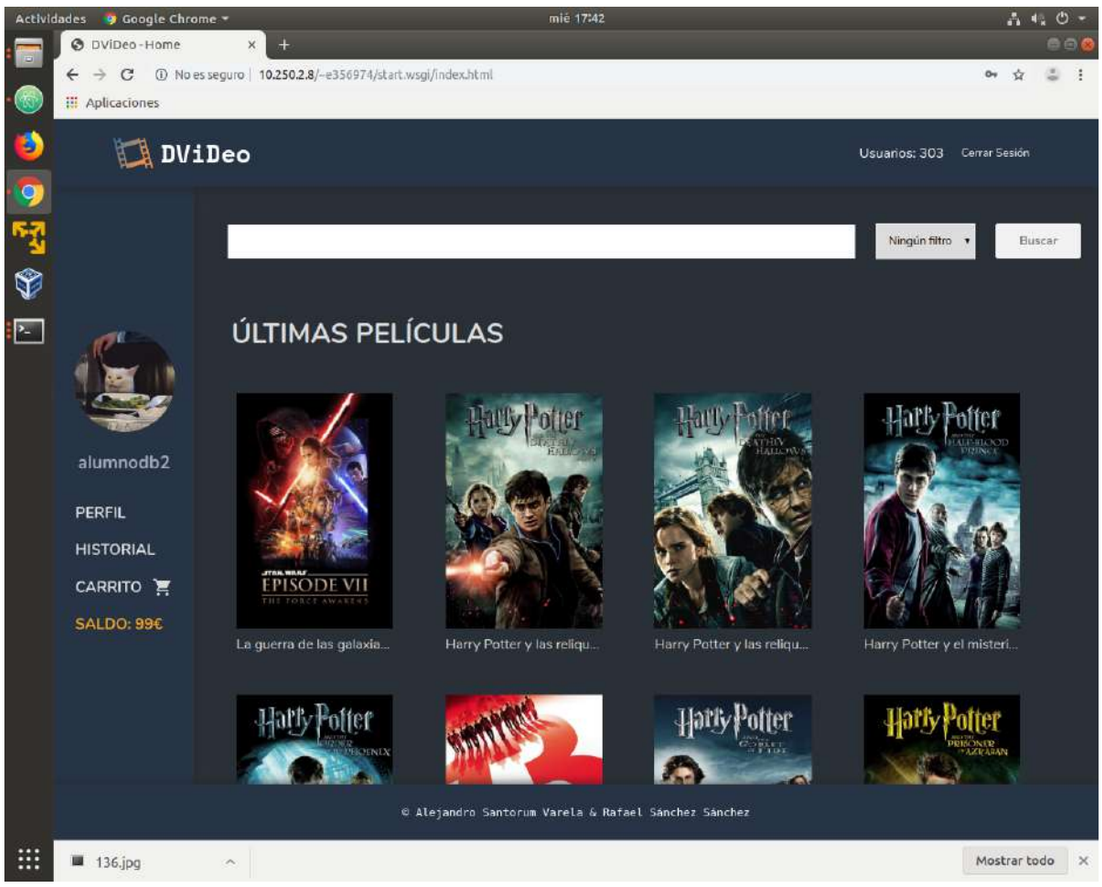


En la imagen anterior se muestra el formulario completado de forma adecuada. Presionando en "Registro" se nos abrirá la página principal de nuevo, donde podremos ir a la página de inicio de sesión (`login.html`):



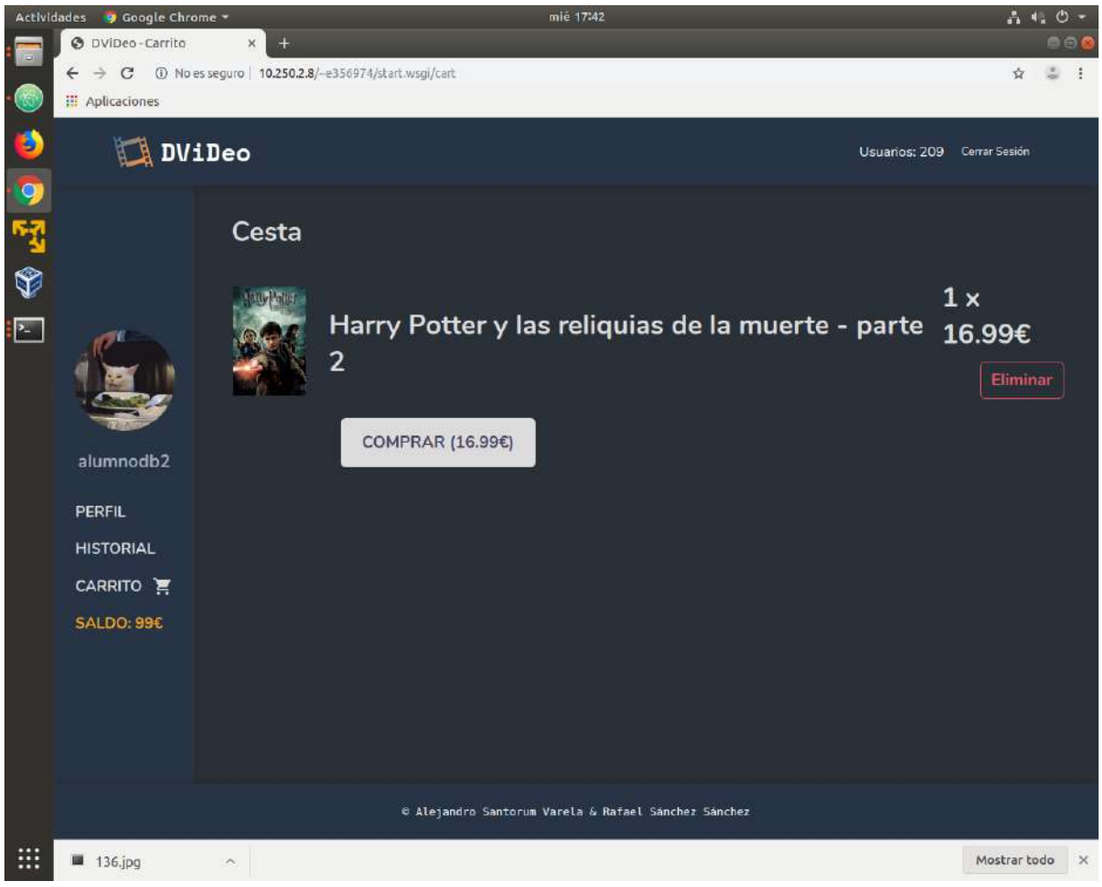
Aquí podremos introducir nuestros datos y *logearnos*. Por razones de simplicidad, la imagen anterior ha sido tomada después de iniciar sesión y cerrarla para mostrar el correcto funcionamiento de las *cookies*, ya que el campo del nombre de usuario ya se encuentra prerelleno.

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.

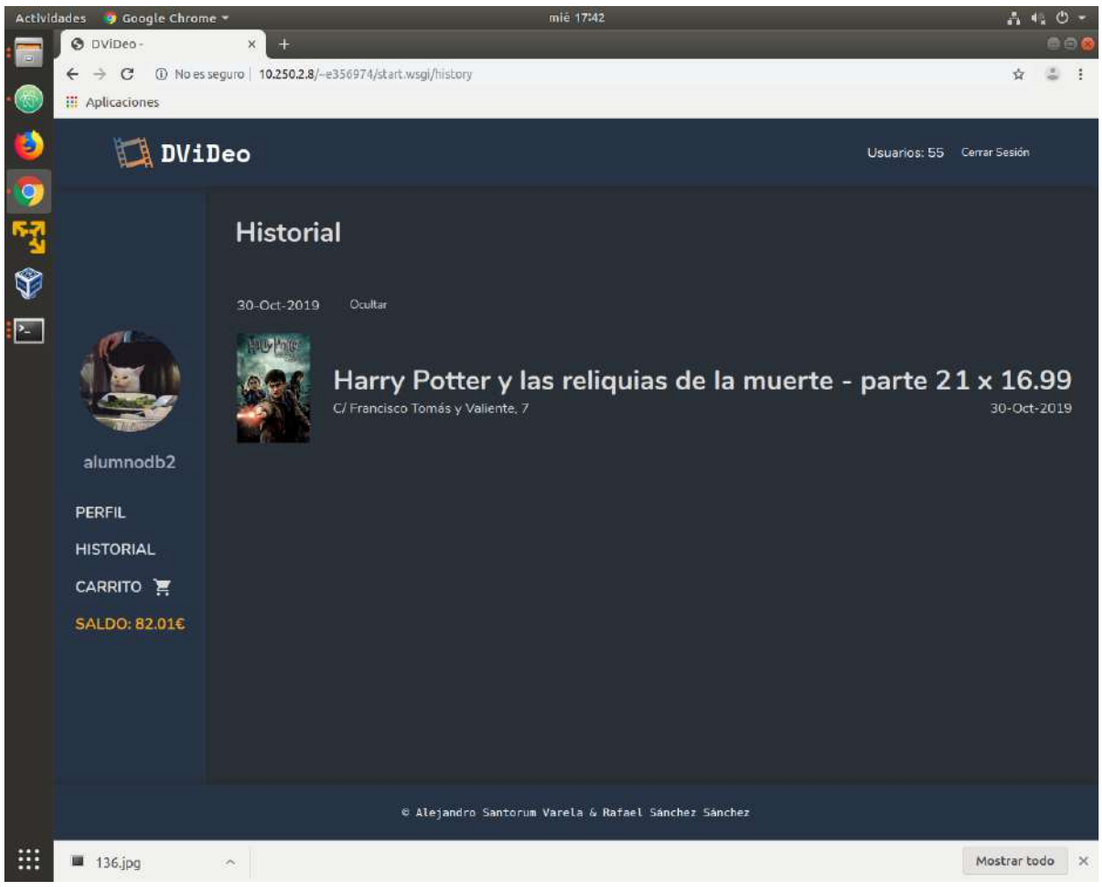
En este momento ya estamos listos para realizar la compra de los productos que habíamos añadido:



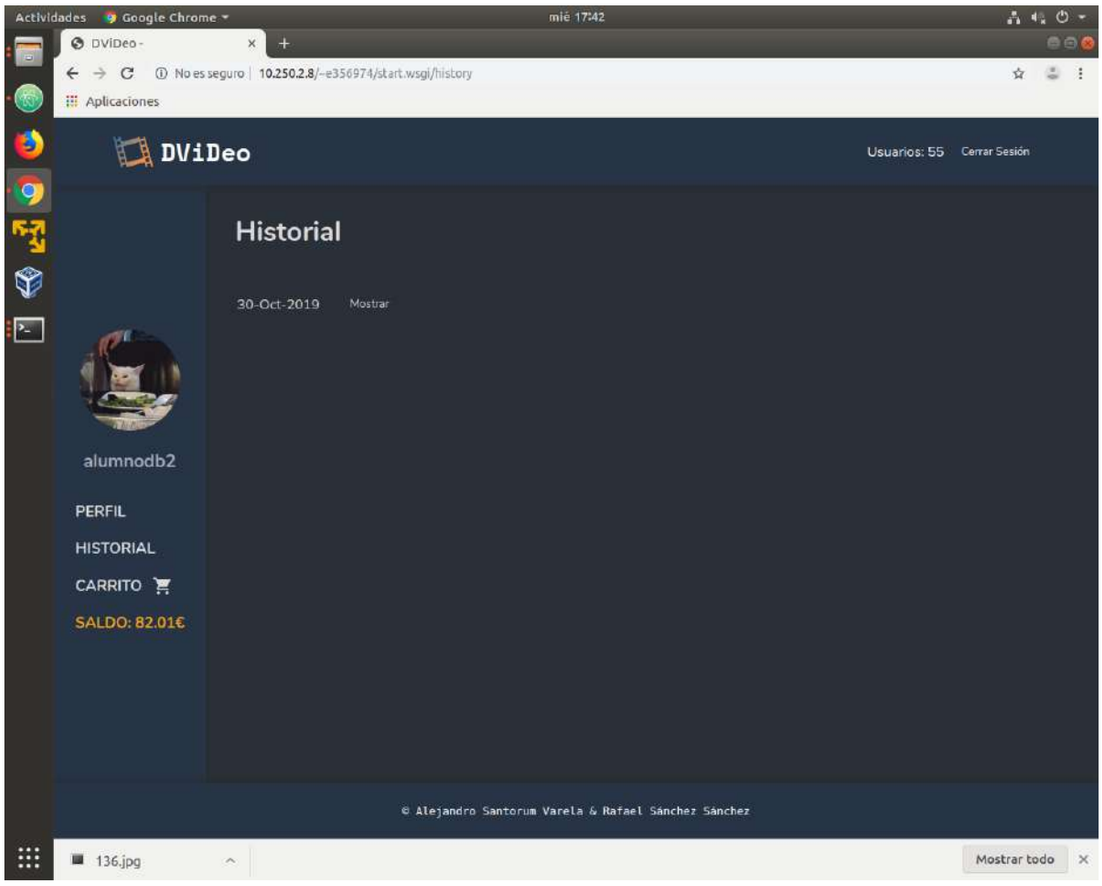
En la imagen anterior se pueden ver los productos que habían sido añadidos antes incluso de iniciar sesión, viendo que el funcionamiento de las sesiones de Flask y el guardado de datos

en los ficheros del usuario ha sido el esperado.

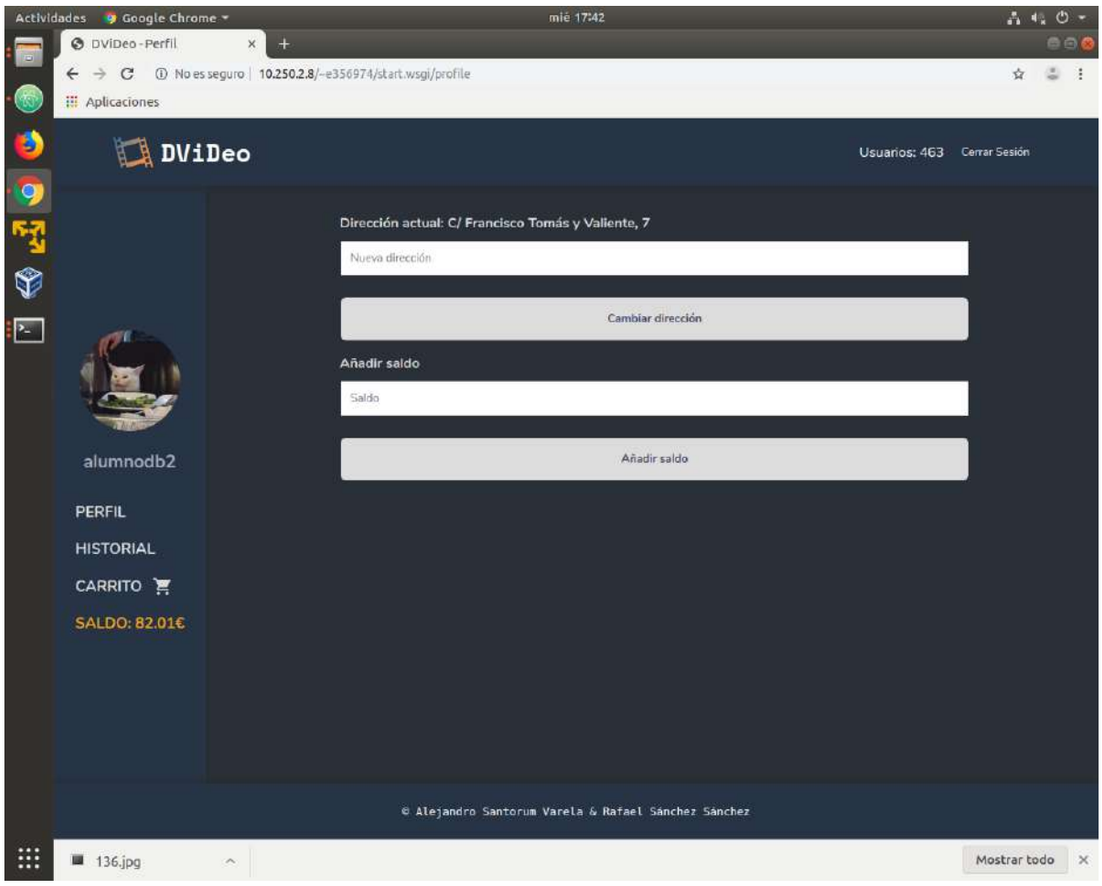
Finalizamos la compra pulsando en "Comprar" y si tenemos el saldo suficiente se nos redirigirá a la página principal con el saldo e historial ya actualizado. Esto último lo podemos comprobar pinchando en "Historial" en el menú lateral (`history.html`):



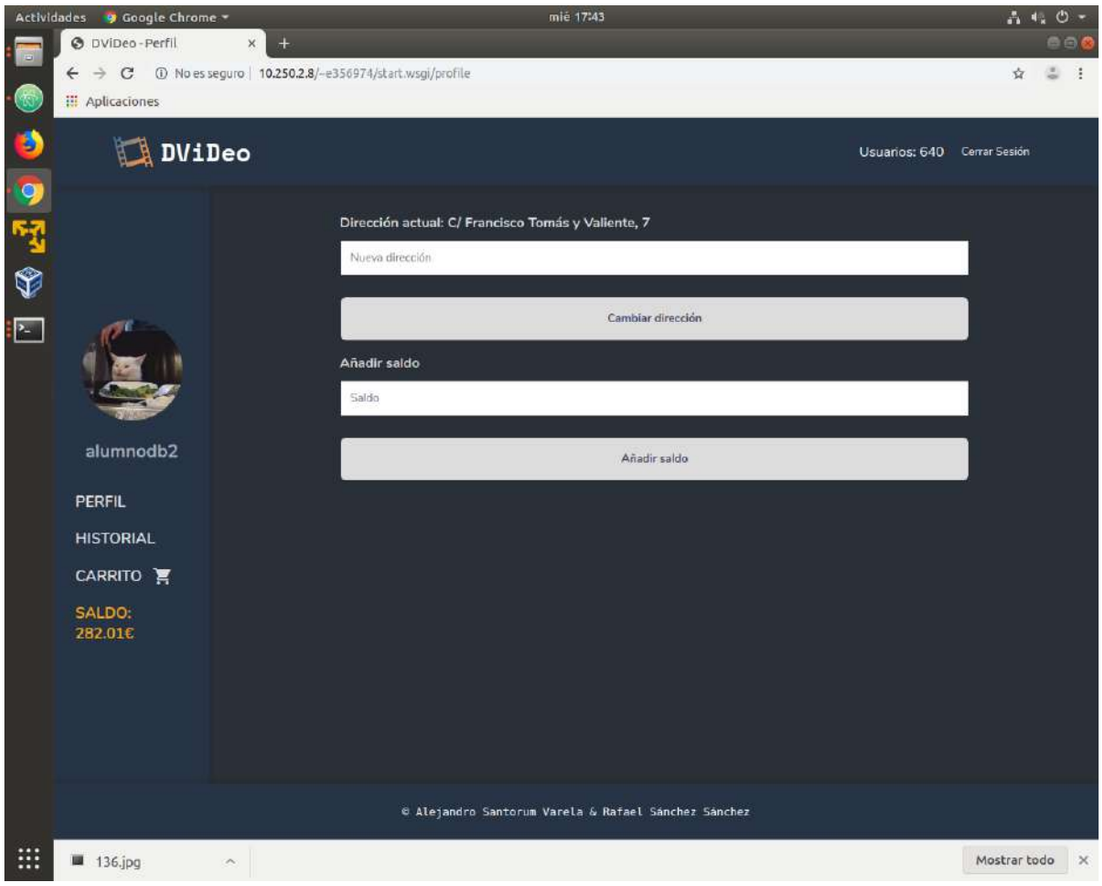
Se puede comprobar que en el historial se incluye la compra realizada recientemente, con la fecha, cantidad, precio unitario y dirección de entrega. Además, se aporta un botón por el cual podremos ocultar la lista de productos comprados en una misma fecha:



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.

8 Conclusiones

Llegado el final de esta práctica es el momento de recapitular. En la primera práctica se ha desarrollado el *frontend*, en la segunda el *backend* utilizando el *framework* de Python `Flask` y en esta hemos implementado la base de datos en PostgreSQL, bastante más profesional que en un simple fichero JSON.

Ahora nos centramos ya en la siguiente, donde nos encargaremos de realizar las posibles funcionalidades que aún faltan y en mejorar la seguridad de nuestra aplicación.

9 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/`
- PostgreSQL `https://www.postgresql.org/`
- SQLAlchemy (`https://www.sqlalchemy.org/`)