Práctica 1. Programación Web dinámica de un sistema de venta de DVDs

Versión 2.1 14/09/2021

Objetivos docentes

Tras la realización de la Práctica 1, el estudiante debe ser capaz de:

- Maquetar documentos HTML que incluyan divisiones/secciones (div), tablas, formularios, imágenes, etc. Se valorará el uso de *tags* específicos de HTML 5.
- Definir el aspecto visual de los documentos mediante ficheros de estilo CSS.
- Entender la diferencia entre abrir un documento accediendo directamente a sus ficheros desde el navegador o a través de un servidor web.
- Comprender el papel de los lenguajes de script en la interfaz de usuario para sistemas Web.
- Conocer los elementos fundamentales del lenguaje Python para implementar funcionalidad del lado del servidor.
- Implementar programas sencillos con Python y Flask (micro-framework para desarrollar páginas web en Python): procesamiento de formularios y generación dinámica de la página, persistencia de sesión y mantenimiento de estado, etc.

Enunciado

Los directores de una empresa de venta de películas (DVD, Blu-ray, etc.) necesitan actualizar su página de ventas por internet. Para ello, deciden contratar con una empresa de consultoría una interfaz de usuario basada en Web para facilitar la venta de sus productos por este medio. En la metodología que han seleccionado para el desarrollo del proyecto, los directores desean disponer de distintas entregas en las que puedan ir valorando los resultados del proyecto, interactuando con la empresa seleccionada para hacerle llegar comentarios. Por ello, en esta **primera entrega**, los directores desean que se dé cumplimiento a los siguientes requerimientos:

- Se realizará en lenguaje **HTML** con uso de CSS, usando Python del lado del servidor.
- Se deben **usar rutas relativas**, no absolutas, para referenciar enlaces a otras páginas.
- El ***layout*** de las páginas se deberá realizar mediante **divisiones (div) o** las nuevas funcionalidades de **HTML5 (header, footer...) y no con tablas**, que se reservarán para mostrar información tabular.
- Todas las páginas de la aplicación web se deben estructurar del mismo modo y tener los siguientes elementos comunes: una **cabecera**, un **menú lateral**, un **pie** de página y una zona de **contenido**. El contenido variará en cada página. Se podrá hacer uso de *iframe*'s para evitar copiar y pegar elementos comunes.
  Ayuda: Revisar la documentación de HTML y CSS en la W3 schools (https://www.w3schools.com/), en particular aquella relativa al elemento CSS3 Flexbox. *Implementación avanzada*: Se valorará el mantener la cabecera y el pie siempre visibles, sin desaparecer al hacer *scroll*.

- No se permite el uso de imágenes de fondo. El fondo ha de ser de color plano o degradado. Pero sí **se deberán usar imágenes** para ilustrar elementos de la web como, por ejemplo, las distintas películas del catálogo, el logo de la web, etc. Se valorará además la existencia de imágenes que sean *clickables*.
- Se valorará el uso de **elementos flotantes** (uso de selector ***float***, display, … de CSS).
- El estilo del sitio se deberá realizar con un **fichero** común de estilo (**CSS**) para todas las páginas. En principio, no se deberá introducir ningún modificador de estilo dentro del HTML ni como hoja de estilos interna, ni *inline*. Si se utilizan estas modalidades se deberá justificar en la memoria.
- En ningún caso se permite introducir estilo mediante los atributos de estilo de HTML.
- Todas las páginas creadas deberán tener la etiqueta <!DOCTYPE> adecuada a HTML 5, y se deberán validar para esta versión de **HTML** utilizando el **servicio de validación** de W3C: http://validator.w3.org
- Adicionalmente, el fichero de **estilo** de deberá validar utilizando el **servicio de validación** de W3C: http://jigsaw.w3.org/css-validator/. También se puede utilizar http://csslint.net/
- La aplicación no requerirá vía enlaces externos de ningún material (por ejemplo, imágenes) que no se encuentre en el propio servidor.

- La funcionalidad dinámica de las páginas se realizará en Python haciendo uso de Flask y, **como mínimo**, de la siguiente funcionalidad:
    - Mezcla de código HTML y Python.
    - Uso de diccionarios y arrays en Python.
    - Estructuras de programación (if, bucles, etc.).
    - Funciones de Python.
    - Persistencia de información de la sesión.

- Las páginas no accederán a los contenidos de una base de datos relacional, sino que los datos serán almacenados en ficheros JSON. En particular, se creará un fichero, `historial.json`, uno por cada usuario, que se actualizará automáticamente a partir de las compras que realice el usuario.

- La aplicación web se ejecutará en el servidor local Apache (*http://localhost/~`usuario`*), publicando los contenidos en el directorio */home/`usuario`/public_html*.

  Nota: El servidor Apache debe tener instalado el paquete **mod_wsgi**, que permite ejecutar cualquier aplicación web implementada en Python. Consultar el ejemplo en 'Instrucciones para Implantar la Plataforma' que hay en Moodle.

- En esta entrega, la información simulada se encontrará en un fichero JSON (`catalogo.json`).
  Ayuda: Un fichero JSON que se puede estructurar; por ejemplo:

```
{
   "peliculas": [
      {
         "id", 1,
         "titulo": "La guerra de las galaxias: Episodio 1-La amenaza fantasma",
         "poster": "imagenes/guerraepisodio1.jpg",
         "director": "George Lucas",
         "precio": 18.99,
         "categoria": "Aventura",
         "anno": 1999,
         "actores": [
           {
              "nombre": "Jake Lloyd",
              "personaje": "Anakin Skywalker",
              "foto": "imagenes/anakin.jpg"
           },
           {
              "nombre": "Ewan McGregor",
              "personaje": "Obi-Wan Kenobi",
              "foto": "imagenes/kenobi.jpg"
           }
         ]
      }
   ]
}
```

Nota: Nótese el uso de arrays, por ejemplo 'actores': debe evitarse crear campos del estilo de 'actor1', 'actor2', ...

Consejos sobre el uso de Flask

1. Se suministra un ejemplo de aplicación web en el archivo `example-web-v<n>.zip`. Se recomienda revisarlo detenidamente.

2. Este ejemplo muestra la estructura típica de una aplicación web Flask que, en nuestro caso, se desplegaría en el directorio ~/public_html:

1. `start.wsgi`: punto de entrada cuando se despliega bajo Apache, usando WSGI
2. `app/`: aplicación web
3. `app/__init__.py`: código de inicialización y que indica que es un módulo python
4. `app/__main__.py`: para poder ejecutar la aplicación en modo depuración con el mandato `python3 -m app`
5. `app/*.py`: código asociado a las distintas URL de las páginas de la aplicación Flask (modelo y controlador)
6. `app/templates/`: plantillas jinja2 de páginas web HTML de la aplicación (vista). Opcionalmente, se pueden organizar en *blueprints* y *views* (ver el tutorial de Flask en la bibliografía)
7. `app/static/`: ficheros auxiliares de la aplicación web, como esitlos CSS, código javascript (por ejemplo en `app/static/js`), imágenes (por ejemplo en `app/static/img`), etc.

3. El ejemplo suministrado funciona tanto bajo el servidor Apache (instalándolo en public_html y en la url http://localhost/~<usuario>/start.wsgi) como en modo depuración, lanzándolo desde el terminal con los mandatos:

```
source si1pyenv/bin/activate
python3 -m app
(se requiere el entorno python si1pyenv descrito en las 'Instrucciones para Implantar la Plataforma')
```

4. Usad `url_for` (https://flask.palletsprojects.com/en/1.1.x/api/?highlight=url_for#flask.url_for), tanto en el código python como las plantillas html de Flask, para obtener las rutas de los distintos elementos de la aplicación. De esta forma, la aplicación funcionará tanto en el entorno de depuración en terminal, mediante el mandato `python3 -m app`, como en el entorno de producción, bajo Apache.

5. Si se altera alguno de los elementos de la sesión, se ha de establecer: `session.modified=True` (https://flask.palletsprojects.com/en/1.1.x/api/?highlight=session%20modified#flask.session.modified). La modificación de objetos de la sesión no siempre se registra automáticamente bajo Apache.

6. Usar la variable `app.root_path` al abrir (`open`) ficheros de la aplicación. Ver `routes.py`.

7. Para recoger los valores enviados en un formulario, haced uso de la variable `request` (https://flask.palletsprojects.com/en/1.1.x/api/#incoming-request-data). Los datos enviados se encuentran en:
   - el campo `request.form` si son enviados con el método POST
   - el campo `request.args` si son enviados con el método GET

- el campo `request.values`, que contiene los datos enviados por cualquier método

8. Al ejecutar la aplicación web bajo Apache y WSGI, los errores y mensajes de depuración se pueden consultar en `cat /var/log/apache2/error.log`

9. Recordad que con F12 se accede al depurador de Firefox y Chrome, y que se puede forzar un refresco duro de la página con Ctrl-F5. Puede que al redesplegar la aplicación bajo Apache y WSGI el navegador no detecte los cambios; se recomienda entonces abrir una ventana en modo incógnito.

Funcionalidad

Es imprescindible que todas las páginas indiquen si se está navegando con un usuario o no. En el caso de que el usuario ya se haya registrado aparecerá el nombre de usuario. En caso de que no se haya registrado debe aparecer un enlace para hacer login.

Todas las páginas deben tener una serie de elementos comunes: una cabecera, un menú lateral, una zona para mostrar los contenidos y un pie de página.

Entrada: Página Principal

La página muestra en la zona de contenidos una selección de DVDs disponibles, así como una caja de texto para realizar búsquedas y un desplegable para filtrar por categoría. La funcionalidad de búsqueda y filtrado se implementarán mostrando una página en función de la categoría y el texto insertado por el usuario, usando diccionarios con datos precargados en Python.

Los datos para esta página se tomarán del fichero `catalogo.json`, que contendrá todas las películas disponibles. El filtrado de esos datos se debe realizar mediante Python.

Esta pantalla también contendrá un acceso a la cesta o carrito, que contiene los artículos que el usuario quiere comprar, pero que no ha pagado aún.

Notas de implementación: En esta página, entre otros elementos, se podrá practicar la combinación de ***div* y *float*** para colocar elementos (por ejemplo, últimas películas) que se recoloquen al redimensionar la ventana del navegador. También conviene hacer uso de **img** para incorporar imágenes, que además deberían contener un enlace a la página de detalle.

Página de acceso

La página de acceso deberá comprobar que el usuario existe y validar la contraseña. Para ello, abrirá el archivo del usuario indicado (ver la siguiente sección "Registro de un Usuario"), se compararán las contraseñas en blake2b, y si coinciden se dará acceso al sistema.

Nota: Se puede hacer uso de la biblioteca de python 'hashlib' y el método 'hexdigest': `hashlib.blake2b((salt+'mi contraseña').encode('utf-8')).hexdigest()`

Se mantendrá, mediante sesión, la información del usuario al ir navegando por el sistema.

Se deberá implementar también la funcionalidad de salir o desconexión de un usuario.

La información del último usuario que inició sesión se mantendrá en una cookie, que se usará para precargar el campo 'usuario' (o similar) del formulario de acceso con este valor.

Registro de un Usuario

Esta página mostrará en la zona de contenido un formulario para pedir datos de un nuevo usuario. Se solicitará al menos la siguiente información: nombre de usuario, contraseña, email y tarjeta de crédito. También habrá un botón para confirmar.

Dentro de la carpeta de la aplicación web (/home/alumnos/*usuario*/public_html) se creará una carpeta de nombre `usuarios` que contendrá una subcarpeta para cada usuario. Dentro de esta carpeta se creará un fichero (`datos.dat`) donde se guardará en variables el nombre del usuario, contraseña (cifrada en blake2b), correo electrónico, número de tarjeta de crédito y saldo. El saldo inicial que se asignará a cada usuario cuando se registre será determinado por un número entero aleatorio entre 0 y 100.

Ayuda: Cuidado con los permisos de la carpeta `usuarios`, ya que la carpeta debe ser accesible por el usuario bajo el que se ejecuta el servidor web Apache (¿cúal es este usuario en Ubuntu?): revisar los mandatos, y funciones de la biblioteca 'os' de python, 'mkdir' y 'umask'.

Esta carpeta propia de cada usuario también contendrá el historial de compras (fichero `historial.json`).

La página de registro deberá comprobar si el usuario ya existe. Para ello, comprobará si existe una subcarpeta con el nombre del usuario. Si es así, generará un error diciendo que el usuario ya existe. En caso contrario, continuará con el registro y se crearán la carpeta y el archivo correspondiente.

El formulario de registro debe contar como mínimo con los siguientes elementos:

1. Usuario: El usuario no deberá contener caracteres especiales ni espacios y la longitud deberá ser al menos de 6 caracteres.

2. Email: Correo electrónico del cliente. Debe comprobarse que el email tiene un formato válido.

3. Clave: Contraseña del cliente. En el caso de la clave, deberá tener al menos 8 caracteres.

4. Confirmación: Campo para confirmación de la clave. Se debe comprobar que este campo y el campo clave deben tener el mismo valor. Tanto el contenido de este campo como el de clave deben ofuscarse para que no sean visibles mientras se introducen.

5. Fortaleza de la clave: Se utiliza para mostrar al usuario la fortaleza de la clave introducida y debe actualizarse cada vez que el usuario introduzca un nuevo carácter en este campo. La fortaleza de la clave va a estar definida por distintos factores como longitud de caracteres, números, caracteres especiales, mayúsculas, etc. El valor de la fortaleza deberá ser: debil, normal ó fuerte, dependiendo de los factores previamente descritos.

6. Tarjeta de crédito: Tarjeta de crédito del cliente. Debe estar formada por 16 dígitos.

7. Dirección: Dirección donde se realizará el envío de las compras. Deberá tener una longitud máxima de 50.

Los datos introducidos en el formulario deberán ser validados en cliente. En los casos donde exista un control HTML5 específico con soporte en los dos navegadores principales (Chrome y Firefox), y ese control haga la verificación sintáctica, podrá ser usado. En otro caso, los datos se validarán mediante funciones de JavaScript.

Notas de implementación: En esta página se debería **practicar con la mayor cantidad posible de los tipos de elementos disponibles en un formulario**: texto, entrada de texto, botones, listas seleccionables, etc.

Detalle de película

En esta página se mostrará la película seleccionada, y permitirá añadirla al carrito de la compra.

La información de cada película se extraerá del fichero `catalogo.json`.

Carrito de la compra

Se deberá implementar la funcionalidad de carrito de la compra. Este carrito debe ser implementado utilizando sesiones en **Flask Session,** y debe estar disponible en todas las páginas.

No será necesario que el usuario esté registrado para que el sistema le mantenga el carrito de compra, pero sí para poder confirmarlo: de no estar registrado y querer confirmar el carrito, se pedirá al usuario que se registre.

Se deberá implementar la funcionalidad de añadir un DVD al carrito (en la página de detalle de DVD), actualizar la cantidad y borrar un DVD del carrito, mostrar el contenido del carrito y finalizar la compra.

Finalizar compra de DVD

Finalizar la compra implica solicitar todos los productos incluidos en el carrito. Para ello el usuario debe estar registrado. Si no lo estuviera, se solicitará que lo haga al intentar finalizar la compra.

Como programa de fidelización de clientes, se debe incluir un sistema de puntos de compra. Después de cada compra, el cliente acumulará el 5% de sus compras en puntos, donde cada punto podrá ser canjeado por un descuento en las siguientes compras. Cada 100 puntos corresponden a 1€. Los puntos no tienen fecha de caducidad. Estos puntos se almacenarán en el fichero `datos.dat`.

Antes de finalizar la compra se debe comprobar que el usuario tenga saldo disponible. En caso de que no hubiese saldo suficiente, ninguna compra del carrito se confirma. En caso de que sí disponga de saldo suficiente, el importe del carrito se descontará del saldo y éste se actualizará en el fichero de datos del usuario. De la misma manera, si el usuario selecciona utilizar puntos de compra, se debe comprobar si hay puntos suficientes. En el caso en que tenga puntos suficientes, se debe descontar y actualizar el fichero de datos. En el caso contrario, no se confirma la compra.

Una vez confirmadas y pagadas las compras, los datos de las mismas se añaden al final del fichero `historial.json`.

Mostrar historial de un usuario

Un usuario registrado podrá ver todas las compras realizadas hasta el momento, así como el saldo de su cuenta. También podrá incrementar el saldo disponible, que se actualizará en el fichero `datos.dat` con los datos del usuario.

Para esta práctica se implementará un fichero JSON, `historial.json`, que codificará los datos del historial. Estos datos se accederán y mostrarán utilizando funciones Python.

Notas de implementación: En esta página se debería **practicar con el uso de tablas**.

Otras funcionalidades

- Utilizar jQuery para desplegar/contraer detalles de los pedidos en el historial del usuario.
- Utilizar jQuery para implementar un medidor de la fortaleza de la contraseña, pudiendo incluir una barra de progreso.
- Utilizar AJAX para mostrar un banner con el número (generado aleatoriamente en el servidor llamando a un método de Python) de usuarios conectados al sistema de Venta de DVDs, que se debe actualizar cada 3 segundos.

Acceso al portal vía el servidor Apache

Las páginas del portal deberán instalarse en el lugar adecuado: directorio $HOME/public_html del ordenador, donde $HOME es el directorio del usuario linux (al que se accede ejecutando el mandato 'cd').

La URL a visitar en este caso sería **http://localhost/~*<usuario-linux>***, donde *<usuario-linux>* es el nombre de usuario.

También conviene practicar el acceso al portal del compañero de prácticas, averiguando su dirección IP. Para ello se puede ejecutar el mandato 'ifconfig -a' en el ordenador del compañero para averiguar su IP. La URL a visitar en este caso sería **http://*<IP-compañero>*/~*<usuario-compañero>***.

Documentación a aportar

En esta práctica deberéis entregar:

- Ficheros HTML, JSON, Python y JavaScript (y cualquier otro adicional) necesarios para la ejecución de la práctica.
- **No se deberá entregar el entorno de python si1pyenv.**

Entrega

Esta práctica se dividirá en dos entregas. La primera entrega, si bien obligatoria, no será evaluada y servirá para ver el progreso del proyecto. El contenido de las entregas será:

- Tercera semana: Entrega parcial con código HTML, CSS y JS
- Quinta semana: Entrega final con la práctica completa

La fecha y normas de entrega de las prácticas se encuentran en Moodle.

Bibliografía

[1] Aprendizaje interactivo de Python: http://www.diveintopython3.net/

[2] Uso de JSON en Python: https://docs.python.org/2/library/json.html

[3] Tipos de datos: https://docs.python.org/2/tutorial/datastructures.html

[4] Aprendizaje interactivo de JavaScript: https://www.w3schools.com/js/

[5] Flask tutorials:
https://www.tutorialspoint.com/flask/index.htm
https://flask.palletsprojects.com/en/1.1.x/tutorial

[6] Flask cookies: https://www.tutorialspoint.com/flask/flask_cookies.htm

[7] Sesiones con Flask-Session: https://pythonhosted.org/Flask-Session/

[8] WSGI: https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

[9] WSGI con Flask: http://flask.pocoo.org/docs/0.12/deploying/mod_wsgi/

[10] WSGI con virtual environment: https://modwsgi.readthedocs.io/en/develop/user-guides/virtual-environments.html