UNIVERSIDAD AUTÓNOMA DE MADRID

SISTEMAS INFORMÁTICOS I

(2019-2020)

PRÁCTICA 2

Alba Ramos Pedroviejo
Javier Lozano Almeda

Grupo 1363

Madrid, 3 de Noviembre de 2019

ÍNDICE DE CONTENIDOS

- Descripción de ficheros y funcionamiento
  - routes.py: la lógica del servidor
    - index
    - detalle
    - history
    - about
    - carrito
    - login
    - signup
    - logoutdic
    - itemInDictIndex
    - getCartTotal
    - generateRandomInt
    - setcookie
    - getcookie
  - base.html
  - index.html
  - detalle.html
  - about.html
  - login.html
  - signup.html
  - carrito.html
  - historial.html
  - javascript.js
- Fuentes consultadas

Descripción de ficheros y funcionamiento

routes.py: la lógica del servidor

En este fichero hemos implementado toda la lógica sobre las rutas de nuestra aplicación. Cuando se llame a cada ruta, se va a gestionar la petición de diferentes maneras según cada caso, ya que las rutas pueden solicitarse usando un formulario (método POST) o de otra forma. A continuación detallaremos cada uno.

index

Esta es la ruta principal, a la que te lleva si no se pone ningún argumento. Gestiona el catálogo de películas que mostrará la página, que bien puede contener todas las películas de nuestro JSON (en el caso de que se acceda a la página normalmente), o bien muestra algunas de ellas si se ha recargado la página tras buscar en el buscador. En este caso, filtraremos ese diccionario de películas y guardaremos en una lista auxiliar todas las que encajen con la categoría y título especificados en el buscador (si la categoría elegida es Todo, se filtrará solo por el título). Estos parámetros se obtienen del formulario, mediante request.form y especificando el nombre del campo de dicho formulario. Para realizar el filtro, recorremos el catálogo y comparamos cada entrada con lo que solicita el usuario. Esta comparación se realiza en minúsculas (la búsqueda es case insensitive) y eliminando los posibles espacios al inicio y al final del texto que puedan haberse introducido. Para comprobar si el texto coincide con el título de la categoría, se ha usado la función rfind, que devuelve -1 si la cadena no contiene a la subcadena especificada como argumento.

Finalmente, se cargará la página html correspondiente, a la que se le envía por parámetros el título a mostrar en la pestaña del navegador y el título a mostrar en el bloque principal de la página (que en algunas páginas será el mismo que el anterior y en otros no).

detalle

Esta es la ruta de la página se mostrará la información de una película, que deberá incluir en la url el id de la película deseada. Cada vez que se reciba una petición get, se genera el catálogo a partir del json y se busca en él la película con el id recibido. A continuación se generará la página de detalle con la información de la película. Si se recibe una petición post, se comprobará si el artículo ya estaba en el carrito. Si el articulo no esta en al carrito, se añadirá a session[carrito] y se mostrará la página con una alerta indicando que el artículo se a añadido al carrito. En caso de que el artículo ya se encontrase en el carrito, se aumentará en session[carrito] la cantidad de dicho producto y se mostrará la página con una alerta indicando que el producto ya estaba y que se ha aumenta la cantidad de dicho producto.

history

Esta ruta lee de sessions el usuario, que se encuentre logueado en ese momento, busca en su directorio el json del historial y los carga en un diccionario, a continuación se genera la página del historial pasándole como información el diccionario con el historial y el catálogo completo.

about

Esta ruta carga la página de información sobre los creadores sin ningún aspecto destacable.

carrito

Si se accede a esta ruta usando el método get, se genera la página del carrito enviando la información que contenga session[carrito]. En caso de que se reciba una petición post se realizarán diferentes acciones dependiendo del contenido incluido en el form, las acciones a realizar son:

- Si el formulario no incluye el campo 'cartAction', es que se quiere realizar un cambio en la cantidad de algún ítem. Para ello se leerán del form el id del artículo y la cantidad y se modificará en session[carrito]. tras ello se generará la página con los nuevos datos del carrito.
- Si el formulario si que incluye el campo 'cartAction', se comprobará la acción indicada en este. Dependiendo de la acción solicitada se procederá a:

    - Si la acción es 'delete', se borrarán de la lista los ítems incluidos en ‘selectedItems’ y se generará la página con los nuevos datos del carrito.
    - Si la acción es 'comprar', se comprobará si hay algún usuario ha iniciado sesión. Si no se hubiera iniciado sesión se mostrará un mensaje de alerta indicando que se debe de iniciar sesión. si la sesión sí que estubiera iniciada, se leerá del archivo de datos del usuario el saldo disponible y en caso de que este fuera disponible se modificara en el archivos de datos el nuevo saldo, se añadirá la compra al historial del usuario y se vaciara el carrito. en caso de que no hubiera saldo suficiente se mostrará un mensaje de alerta.

login

A esta ruta se puede acceder de dos métodos diferentes; con get para mostrar el formulario y con post para comprobar la corrección de los datos. Si se accede mediante el método get, cuando se pulse en el botón de login, se mostrará la página con el formulario.
En caso de que lo que se reciba sea una petición post, es decir que el usuario a introducido sus datos en el formulario y a pulsado a enviar, Se comprobará si el usuario existe buscando en el directorio de usuarios una directorio con el nombre solicitado. En caso de que no exista dicho directorio se le reenviará a la página de login mostrando un mensaje de

error. En caso de que el usuario exista, se comprobará si la contraseña recibida es correcta. Si la contraseña fuera incorrecta, se le reenviará a la página de login mostrando un mensaje de error. En el caso de que la contraseña sea correcta, logueamos al usuario añadiendo el usuario a la sesión y la establecemos como modificada, se llama la función setcookie. Esta se encargará de generar la cookie y redirigir al index.

signup

Esta ruta puede solicitarse de dos formas: cuando el usuario pincha el botón de Registrarse, en cuyo caso se le debe mostrar el formulario correspondiente, o bien cuando un usuario ha rellenado el formulario y se envía una petición al servidor para gestionar el registro. En este segundo caso, lo primero es comprobar que el nombre de usuario especificado no exista ya en nuestro sistema. Para ello, vamos a buscar en el directorio usuario (dentro del actual, app) si existe un directorio con ese nombre de usuario (mediante isdir y exists), en cuyo caso significa que ya existía y se recarga el formulario (conservando ciertos campos para mayor comodidad del usuario) mostrando un error informando de esta situación. Si no existe, se obtiene el resto de información del formulario y se crea un nuevo directorio para ese usuario. Este contendrá dos ficheros: un JSON con su historial de compras y un .dat con sus datos. Hemos utilizado with open as file porque cierra el fichero automáticamente tras la escritura. La contraseña la hemos almacenado cifrada en md5 mediante hashlib.hexdigest.

Finalmente, hemos decidido que, si todo sale bien, el usuario se le loguea automáticamente en vez de redirigirle a la página de login. Para ello, añadimos el usuario a la sesión y la establecemos como modificada. De esta forma, las páginas que carguemos y que hereden de base.html detectarán que hay un usuario logueado y mostrarán en la cabecera su nombre y la opción de cerrar sesión. Tras este login, se llama la función setcookie. Esta se encargará de generar la cookie y redirigir al index.

logoutdic

Esta página elimina al usuario de la sesión y la establece como modificada, tal y como hemos hecho en la ruta signup. Tras ello, también redirige a la página principal. No hace falta comprobar que esté logueado cuando intenta cerrar sesión, ya que desde el propio html gestionamos que esta opción no aparezca a no ser que el usuario esté logueado.

itemInDictIndex

Esta función recibe una lista, el nombre de un campo de dicha lista y una string con el dato a buscar. se comprueba por todos los elementos de la lista i la búsqueda se encuentra en el campo seleccionado. En caso de encontrar el elemento solicitado, se devolverá su index. En caso de no encontrar el elemento solicitado, se devuelve -1.

getCartTotal

Esta función recibe un catalogo y calcula el coste del carrito a partir de los productos de este. se devuelve el total en formato float con dos decimales

generateRandomInt

Esta función genera un número aleatorio entre 0 y 10 para simular el número de usuarios conectados al sistema. Es solicitada por el objeto XHTTP de base.html usando AJAX, como se explicará más adelante.

setcookie

Esta función recibe un nombre de usuario y generará una respuesta para el cliente incluyendo una cookie con el usuario recibido y que redirigirá al index.

getcookie

Esta función lee las cookies del navegador y obtiene el usuario, en caso de que este exista. Es solicitada por el objeto XHTTP de base.html usando AJAX, como se explicará más adelante.

base.html

Esta página define los elementos comunes que tendrán todas las demás páginas, que los heredarán de esta. En ella definiremos la cabecera, el pie de página y las barra de navegación tanto superior como lateral, cuyo funcionamiento explicamos ya en la práctica anterior. También será la encargada de importar en el header todos los ficheros que vayamos a utilizar (JQuery, CSS, Javascript...).

Dentro de esta página HTML insertamos bloques en Jinja2 que nos permitirán definir su contenido particular dentro de cada una de las otras páginas. Además, aquí comprobaremos si el usuario que abre las páginas está logueado o no mediante el uso de session. De esta forma, mostraremos en la cabecera un botón para registrarse e iniciar sesión si el usuario no estuviera logueado, o su nombre y la opción de cerrar sesión en caso contrario.

En esta página y en todas las demás que necesitemos redirigir al usuario a otras páginas, necesitaremos indicar la ruta donde se encuentra esa página. Para evitar hardcodear las rutas, usaremos url_for, una funcionalidad de Jinja2 que nos permite indicar solamente el nombre de la página a la que queramos redirigir. Esta se encarga de mandar la petición al servidor, donde está implementada la lógica a seguir para cargar cada página.

En esta template, que tiene el header que muestran el resto de páginas, implementaremos con AJAX un banner que muestre el número aleatorio de usuarios conectados al sistema cada 3 segundos. Para ello, hemos reservado un lugar en el header que por defecto muestra que hay 0 usuarios y que, en cuanto la página se carga, llama a la función de Javascript startTimer, que cada 3 segundos llama a la función actualizarContador. Esta realiza una petición AJAX mediante un objeto XHTTP a la url /rand (que simplemente genera un aleatorio entre 0 y 10), y cuando el callback del xhttp se activa, el resultado aleatorio se muestra en el header mediante la propiedad innerHTML del párrafo contenedor.

También se muestra, en la parte superior derecha de la parte central, los botones de carrito e historial o solo el carrito, según el usuario esté logueado o no. Para ello se ha usado la variable session para comprobar si hay un usuario logueado o no. Para mostrar las imágenes se ha usado url_for, para evitar hardcodear las rutas de las imágenes, simplemente indicamos que se encuentran dentro de static/image (carpetas que ya no importa dónde estén ubicadas, url_for las busca por nosotros).

index.html

Esta página muestra el catálogo de películas existentes en el sistema. Para ello, mediante un bucle con Jinja2 recorre el diccionario recibido del servidor que contiene las películas y sus datos. Permite hacer click en cada película para obtener más información, redirigiendo a la página de detalle, a la que envía como parámetro el id de dicha película.

En la parte superior izquierda hay un buscador que permite filtrar películas por categoría. Mediante un desplegable se muestran las categorías, y el usuario especifica el título a buscar o algo que contenga el título (sin importar las mayúsculas). El servidor filtra el catálogo y recarga la página devolviendo solamente las películas que encajen con lo buscado.

detalle.html

En esta página se muestra toda la información correspondiente a una película. Dicha información será recibida en la variable movie, mediante el uso de html y jinja2 se irán extrayendo cada uno de los atributos de la película e incluyéndose en el html. A parte de la información de la película, se muestra un botón que nos permitIrá añadir al carrito dicha película. Cuando el usuario pulsa en el botón de añadir película, se enviará una petición post al servidor que incluye el id de la película deseada. Una vez que se recibirá la respuesta del servidor que incluirá un mensaje que será mostrado a través de una alerta, usando la función showAlert() incluida en el javascript, en el navegador. este mensaje indicará si el articulo se a añadido al carrito o o si se han aumentado el número de unidades en caso de ya existir dicho artículo,

about.html

Página en la que se muestra una foto de nosotros y se da una pequeña información sobre nosotros y las motivaciones para la creación de este sitio.

login.html

Esta página muestra un formulario para que el usuario inicie sesión. Una vez se cargue la página , se comprueba si el usuario ya se había logueado anteriormente comprobando la existencia de nuestra cooki mediante la función “getcookie()”, esta función realiza una petición AJAX mediante un objeto XHTTP a la url /getcookie (que simplemente comprueba la existencia de la cookie y en caso de que exista devuelve el nombre de usuario), y cuando el callback del xhttp se activa, el nombre de usuario se muestra en el campo del nombre de usuario mediante la propiedad value de dicho input . Como los datos de los usuarios se encuentran exclusivamente en nuestro servidor, las comprobaciones se harán en este mediante un post al pulsar en enviar. Una vez enviada la petición se recibira la respuesta del servidor, que puede resultar en un mensaje de error si el usuario no existe o si la contraseña es incorrecta o en iniciar la sesión del usuario y redirigirlo al index.

signup.html

Esta página muestra un formulario para que el usuario se registre, cuyos datos se envían al servidor con el método POST. Cada campo del formulario debe cumplir las normas especificadas en el enunciado de la práctica, y esta comprobación se realizará en cliente para evitar sobrecargar el servidor con los registros incorrectos de, posiblemente, cientos de usuarios en una aplicación real. Para validar la información hemos utilizado mayoritariamente HTML5.

No se permitirá que ningún campo esté vacío: esto lo hemos comprobado en el propio tag utilizando el atributo pattern, que permite definir, mediante expresiones regulares, un mínimo o un máximo de caracteres, o el número exacto requerido para ese campo. Con pattern también hemos definido los caracteres permitidos en cada campo. Por ejemplo, para el nombre de usuario, solo permitimos letras minúsculas (a-z) o mayúsculas (A-Z) o números (0-9). HTML5 también permite especificar el tipo de dato que se introducirá en cada campo. Así, para las contraseñas definiremos el tipo password, que nos ocultará los caracteres. Además, para el email detecta cuándo no se ha puesto la @, o si no se introduce texto tras ella.

Para el campo que repite la contraseña y comprueba que coincide con la introducida anteriormente, hemos usado Javascript. Esto se debe a que debemos comprobar carácter a carácter con el contenido de un tag distinto, y HTML no nos lo permitía. Para ello, hemos creado una función, checkPasswordCoincidence, que compara el texto introducido en este campo con el del campo anterior y lanza un mensaje de error si no coinciden. En relación

con la contraseña, también se ha creado con JQuery un medidor de la fortaleza de la contraseña. Hemos detectado con JQuery el evento de tecla levantada sobre este campo del formulario. Cada vez que se añada un nuevo carácter a la contraseña, la función del medidor se ejecutará: esta comprobará diferentes parámetros que hemos considerado para medir la fortaleza. Si la contraseña tiene 8 dígitos o más, que es el mínimo, incrementamos un contador. Si, además, combina mayúsculas y minúsculas, se vuelve a incrementar. También si contiene números, caracteres especiales o dígitos (\d). Todo esto se ha realizado con expresiones regulares. El máximo valor del contador puede ser 5, y 5 son los posibles anchos y colores que le daremos a la barra (cuanto más segura sea, más ancha será la barra, y será verde, mientras que si es insegura será roja y pequeña). Finalmente, en el propio código se establece la clase del elemento (div) donde está la barra de progreso, y desde CSS editamos el color de relleno para cada clase.

Finalmente, puede ocurrir que el usuario introduzca bien todos los datos, pero que ese nombre de usuario ya esté registrado. Esto necesariamente se tiene que comprobar en el servidor. En él hemos comprobado si existe una carpeta con ese nombre de usuario (usando isdir y exists), y en el caso de que ya existiera mandamos un mensaje de error al HTML que se mostrará en la parte inferior de la pantalla. Este error, no obstante, deja guardados los campos de nombre y correo electrónico del usuario (atributo required value en el formulario, que se obtiene del servidor), no así la contraseña o la tarjeta bancaria, que hemos omitido por seguridad. El usuario puede cerrar este mensaje si lo desea, y esta funcionalidad se ha implementado con un método Javascript que hace invisible el cuadro de error.

carrito.html

En esta página se muestran todos los artículos contenidos en el carrito. Para ello, mediante un bucle con Jinja2 recorre la lista recibida del servidor que contiene cada uno de los artículos del carrito con su id y cantidad.Esta información se irá añadiendo a una tabla, generando una fila por cada artículo que incluirá: un checkbox para en caso de querer eliminar dicho artículo, su foto, su número de unidades en un spinner que nos permitirá aumentar o reducir el número de unidades de dicho artículo y el coste total del artículo teniendo en cuenta el número de unidades. también se incluye una fila columna incluyendo el coste total del carrito, que será recibido desde el servidor. Cada vez que se pulse en el spinner de un artículo, ya sea aumentando o reduciendo la cantidad, se hará una petición post al servidor incluyendo el id del artículo y el nuevo número de unidades, el servidor nos devolverá otra vez la página de login incluyendo la nueva cantidad y el total actualizado.

Además de la tabla con la información del carrito, tenemos dos botones uno para finalizar la compra y otro para eliminar artículos del carrito. El botón “realizar compra”, es de tipo submit y enviará una petición submit al servidor incluyendo el campo "cartAction" con el valor 'comprar', que hará que el servidor compruebe si se cumplen los requisitos necesarios para realizar la compra y según los resultados nos devuelva la respuesta correspondiente. Por el contrario el botón “borrar” llamara a la función “deleteSelected()”, que recogerá en una lista el id de todas las columnas estén seleccionadas. Una vez generada la lista de elementos, si

esta está vacía se mostrará una alerta indicando que no hay ningun articulo seleccionado y si no, se asignará la lista al value del campo 'selectedItems' incluido en el form de los botones. Una vez asignada la lista se cambiara el value del campo "cartAction" a ‘delete’ y se hará un submit del formulario seleccionando este a través de su id. Una vez que se haya realizado la compra y vaciado el carro, la página detectara que el carro esta vacio y no se mostrará la tabla.

historial.html

Esta página muestra el historial de compras del usuario. Recibe del servidor el historial del usuario y el catálogo completo, se crea una tabla y mediante un bucle con Jinja2 se recorre el diccionario del historial y generando una fila por cada pedido incluyendo el número de pedido, la fecha, el precio y su estado. Adicionalmente se genera una fila en la que con otro bucle de jinja2 generamos una tabla con los artículos de dicho pedido y se oculta mediante la función “controlTogle(id)”. Una vez que la página esté cargada, cada vez que el usuario pulse sobre la línea de un pedido se mostrarán u ocultaran los detalles de dicho pedido.

javascript.js

Este fichero contiene todas las funciones de JQuery y Javascript que usan los demás ficheros. Basta con importarlo en base.html y todas lo heredarán. En él también hemos incluido funciones que puedan usarse en diferentes ficheros con diferentes argumentos, más genéricas.
La función hide recibe un id de un elemento HTML y lo oculta.

Fuentes consultadas

https://www.tutorialspoint.com/python/time_sleep.htm
https://api.jquery.com/toggle/
https://www.w3schools.com/jquery/eff_toggle.asp
https://overiq.com/flask-101/cookies-in-flask/
https://treyhunner.com/2016/04/how-to-loop-with-indexes-in-python/
https://stackoverflow.com/questions/42013067/how-to-access-session-variables-in-jinja-2-flask