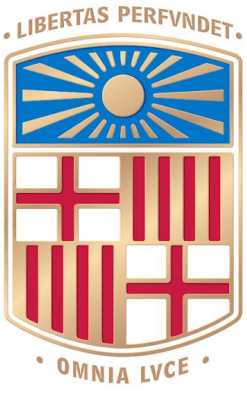

UNIVERSITAT DE BARCELONA

Software Distribuït

Práctica 2: Cliente-Servidor

Joan Seguí Nieto
Jose Manuel Vasquez Chavarria

Objetivo

En esta práctica lo que se nos pide es crear una página web utilizando el framework Django, esta es creada para una multinacional llamada YKEA. El principal objectivo es la creación de la página web con los siguientes puntos. Esta deberá tener una web store donde los clientes podrán buscar entre los distintos productos que ofrece la tienda, añadirlos dentro de una cesta y finalmente comprarlos. Deberá poder mantener diferentes sesiones anónimas para que puedan entrar distintos clientes a comprar en la web store. También tiene que haber un admin para poder gestionar la web store. Finalmente, en la pagina web los usuarios se podrán registrar, entrar (login) y salir (logout).

Nuestra plataforma también ha de soportar una API que sirva como servicio, para poder ofrecer un comparador, esto se podrá hacer creando un usuario comercial que pueda modificar los items. El servicio proveerá búsquedas por categoría, decir si es nuevo no, e ingresar el precio máximo del item que buscamos en el comparador.

Código

Models.py

En el archivo **“models.py”** tenemos todas las clases que necesita nuestro proyecto, estas clases son las tablas que contiene la base de datos de nuestra web a la hora de migrar.

Tenemos la clase **Item** en la cual podemos encontrar todas las características de los items, la clase **Shoppingcart** que haría referencia a la cesta que podemos encontrar en la app store, en esta clase lo que encontramos es el usuario de la cesta y los items que hay en ella, para esto es necesario una clase intermedia que comunique con la clase Item y Shoppingcart en una relación de ManyToMany, la clase intermedia es la que llamamos **Itemquantity**, esta clase está conectada a la cesta y a los items mediante dos claves foraneas, y en la classe Shoppingcart podemos encontrar un atributo items que, con una relación ManyToMany, obtiene los items de la clase Item mediante esta clase intermedia Itemquantity, la clase **Customer** es la que tiene la informació del usuari y el dinero que tiene ese usuario para gastar. En la clase **Bill** podemos ver la cantidad de dinero que hay en la factura de un usuario. Para calcular el total de la factura, esta clase se comunica con la clase **Linebill**, que es la clase donde se calcula la factura en línea y se saca el subtotal de esta. La clase Bill tiene una relación ManyToMany con la clase Linebill, y esta está conectada a la clase Bill y Item con dos llaves foraneas. La clase Linebill es la clase intermedia entre la clase Item y Bill.

Views.py

En el archivo **“views.py”** se encuentran todos los métodos que utilizamos y estos son llamados desde el archivo “urls.py”. En cada método controlamos excepciones relacionadas a la existencia de usuarios y ponemos el carrito correspondiente, exponiendo al usuario la cantidad de items que tiene su carrito. En caso de no estar logeado, no se le muestra nada, solo la opción de logearse o registrarse. El usuario por defecto tendrá 4000€ para comprar items, si supera esta cantidad no podrá comprar.

El primer método que encontramos es el método index, sirve básicamente para enseñar un mensaje de bienvenida al entrar en la web. Luego nos encontramos con varios métodos (**categories**, **items_category**, **item_detail**) que enseñan la información de los items en la página web para que el cliente pueda navegar por esta información.

El método **shoppingcart** sirve para meter en la cesta todo item que exista y que el cliente quiera comprar en ese momento. Básicamente gestiona los items que mete el cliente en la cesta.

Tenemos el método **buy**, este método sirve para comprobar los items que hay en esa sesión y enseñar en la página web estos items que se han iterado, siempre reiniciamos la sesión con la persistencia que tenemos en la base de datos, es decir, los items que están guardados en la base de datos para cada usuario.

Después tenemos un método que hace la función de listener para los botones de delete y checkout. Este método listener es llamado **process**. Y tambíen tenemos dos métodos para los dos botones mencionados anteriormente.

El método **delete** es para que cuando se pulsa el botón de eliminar item, este sea eliminado. Para ello se busca el usuario actual que ha hecho el request y se elimina de la base de datos en la tabla ManyToMany **itemquantity**, y por tanto, también de la sesión.

El método **checkout** sirve para mirar todos los items de la cesta y sacar una factura para el cliente. Para ello se busca el usuario que hizo el request y se crea una nueva factura para este, luego se suman los items que éste tiene actualmente en su carrito y se saca un total, y también si el usuario dispone de dinero puede comprar en caso contrario se le marca un error. Por último para terminar con la factura se agrega el total de los items, si esta factura sobrepasa el dinero del usuario se elimina de la base de datos, en caso de tener dinero suficiente éste se deja y ya no se vuelve a tocar.

El método **register** sirve para que si un usuario nuevo se registra, se compruebe si está bien registrado y este sea guardado con éxito.

También tenemos un método llamado **comparator**. Este sencillamente renderiza y trabaja con un AJAX en el template **comparator.html**.

Finalmente, en este archivo tenemos una clase llamada **ItemViewSet**. Esta clase funciona como filtro de los items.

Todos los métodos que tienen el @login_required, no se pueden usar a menos que el usuario haya hecho el login, en caso contrario se les redirigirá al login.

Serializers.py

En el archivo serializers.py tenemos una clase la cual nos divide los campos del JSON de cada item. Esto lo utilizamos antes de llamar al filtro ItemViewSet.

Templates

Los templates son archivos html y sirven para crear y hacer el funcionamiento de las páginas que aparecen en la web, estan llaman a las urls.py, que contiene los métodos de cada view, para hacer que la página de la web funcione. cada “estado” de la web tiene su plantilla, que es renderizada por las vistas, en casi todas, se pasa información relacionada al usuario y los items, y se comprueba si hay usuario logeado para mostrar unas opciones u otras al usuario.

Testing

Para el test de la página hemos ido probando el estrés que soporta nuestra página, intentando acceder a los puntos críticos, siguiendo las tareas:

- Comprar un item sin estar logeado.
- Registrarse dos veces con el mismo usuario.
- Hacer login añadir al carrito, desconectarse y volver a comprobar que sigue el carrito igual logeandose con el usuario anterior.
- Hacer login y comprar items excediendo el límite de 4000€ puesto por nosotros.
- Hacer login, añadir al carrito, hacer logout, ir para atrás con el navegador y eliminar un item.

Para la API:

- Intentar hacer un PUT mediante **curl** sin logearse.
- Intentar hacer un DELETE mediante **curl** de un item específico sin tener permisos.
- Intentar poner solo el campo category en un request GET para buscar items
- Intentar poner solo un campo de los 3 pedidos en un request GET para buscar items.

Para el comparador:

- Intentar dejar un campo vacío y comparar.
- Intentar comparar items con -1 de precio.

Conclusiones

Con Django como framework, se puede cambiar el paradigma de crear una página web desde 0, ya que es totalmente mantenible y entendible tanto en lo que persistencia de datos se refiere o como reutilización de código de una aplicación, el hecho de reutilizar partes de código creando una aplicación para cada parte de una página web o en nuestro caso, el hecho de reutilizar constantemente plantillas, y renderizarlas con información, hace que la creación de una página web dinámica sea muy rápida de programar.

En esta práctica hemos aprendido que la utilización de Django en las páginas web, o no tan sólo página para usuarios, sino también los módulos **moldeables** que trae como el RESTFUL o el módulo hecho para la gestión de **login**, hace que la extensión de estos sean fáciles y rápidos de hacer de cara a un **SAS**. La seguridad de implementar y evitar explotaciones como XSS son bastante destacables en este framework.