Modelos

En primer lugar, se ha creado un nuevo modelos “VoteBook”, dentro de la aplicación de “Catalog”, cuyos campos son:

- Una clave extranjera o foránea al libro que se desea votar (modelo Book), que en caso de ser eliminado el libro, se elimina estas referencias a los votos también (Delete on Cascade).
- Una clave extranjera a un usuario, que realiza la votación, del modelo de usuarios que ya nos proporciona Django, que, de la misma manera, elimina las referencias de los votos, si el usuario es eliminado de la aplicación.
- Finalmente la puntuación que este usuario establece al libro, siendo un estero del uno al diez establecido con los Max y Min validators.

Por otro lado, una vez tenemos el nuevo modelo, como un libro al principio, se crea sin puntuación (ya que nadie vota cuando el libro se crea), hay que establecer el atributo score a ***null=True***.
Por otra parte, he decidido aunque se que no es necesario, añadir un nuevo atributo en Book, para el número de valoraciones que tiene el libro, que mantendré actualizado, y parte de 0.

Formularios

Una vez tenemos el modelo, ahora hay que añadir el formulario que los usuarios van a tener que rellenar si quieren votar un libro, para ello, en el fichero forms.py de la aplicación de “catalog”, se crea uno llamado “UserVoteBookForm”, y el cual muestra un “TypechoiceField”, con las elecciones (choices) del 0 al 10 para que posteriormente guardemos estas elecciones.

Template

En el template de “book_detail.html”, se crea el nuevo formulario html que es pasado por la vista de book detail (esta vista crea la instancia del formulario creado con anterioridad), que requiere que el usuario esté

autenticado para que se muestra {{ user.is_authenticated }}, y tomara como acción redirigir a la url cuyo valor viene dado por el nombre que comentaremos en el siguiente punto y pasa además el argumento slug del libro, para identificarlo.

Urls

Se añade el nuevo path que acabamos de mencionar, de manera que el anterior formulario nos redirija a la siguiente url:

path("vote_book/<slug>", views.add_vote, name="vote_book")

Esta url que es la que toma acción de haber rellenado el anterior formulario, pasa el slug del libro para saber cual es el libro a votar, y nos lleva a la nueva "view" de "add_vote".

Views

La nueva vista de add_view con los decoradores de @login_required y @require_POST, nos aseguran que a esta url solo pueden llegar usuarios logueados, y que además hemos accedido a través de un POST (a través del formulario que tomaba este método como acción).
Se coge la información del formulario, y se valida. A continuación, se obtiene el objeto "Book" que viene dado por ese campo slug pasado por parámetros, y se obtiene la puntuación que el usuario le ha dado al libro mediante la "cleaned_data" del formulario.
Finalmente, se manda a procesar esta información llamando a una función "add_user_vote", que he creado en un archivo python "vote.py", que se encrgara de realizar toda la lógica de actualización de base de datos.

Pero antes de pasar con ese módulo, mencionar que en la pantalla de home también había que añadir los libros con mayor número de votos, y es por eso que en la vista de "home", he añadido dicha consulta por ese nuevo atributo que cree en el modelo para hacer tracking del número de votos, esta información es mandada al template para ser mostrada.

Vote

Como comentaba, este nuevo módulo se encarga de actualizar los modelos de la base de datos con el nuevo voto añadido, pero tiene la peculiaridad, de que el voto del usuario hacia un libro, es único, de manera que si quiere volver a votar, se debe sobreescribir el voto al que ya había existente.
Es por eso que he encontrado el método “update_or_create”, que hace precisamente eso, para ello, se le pasa el libro a checkear y el usuario, y se añade en el atributo “defaults”, el score al libro. Django se encargará por detrás de ver si tiene un voto anterior o no.

Una vez se ha creado ese voto, ahora hay que recalcular la media de la puntuación que toma ese libro además del número de votos al libro.
Es por ello que se llama a la función “update_book_score”.
Esta función, obtiene los votos de un libro, y suma las puntuaciones de cada uno de ellos.
Finalmente, filtra por el id del libro, y hace un update de estos dos campos, score, que es la suma de todos los votos entre el número de votos, y el número de votos que es el número de votos en cuestión.

Populate

Para poblar, se ha codigo el populate ya existente, y se han creado 100 votos como dice el enunciado.
Para ello se han cogido los usuarios y los libros, y en un bucle hasta 100, se han ido seleccionando usuarios random y libros random para crear una nueva entrada al modelo “VoteBook” con un score random también del 0 al 10.
Finalmente, es necesario llamar al “update_book_score” para actualizar la media de la puntuación a cada libro.

Test junio

A destacar en los tests, ha sido crear un objeto Client() y loguearse en la aplicación para poder acceder a dicho formulario.
A continuación, se ha realizado un post a la url del formulario de votación y se ha pasado como argumento la puntuación. Finalmente, se comprueba en todos los casos que estas puntuaciones van variando.
En cuanto a los test realizados han sido:

1. Probar simplemente una valoración aislada de un usuario, comprobando que esta se ha añadido.
2. Probar la valoración de dos usuarios, de manera que se comprueba que la media es correcta.
3. Probar a repetir la valoración de un usuario, de manera que se vota, se comprueba, y posteriormente se vuelve a votar y se ve el resultado modificado, pero no el incremento de número de votos.
4. Probar a votar con dos usuarios y repetir la votación con uno de ellos, para ver como no se incrementa a más de dos el número de votos, y se recalcula la media bien.
5. Probar a votar con dos usuarios y repetir las votaciones con los dos, de manera similar a la (4), pero repitiendo las dos valoraciones.