(Attention, document en mode édition, ne rien modifier svp)

Cahier des charges

Résumé

L’application permet de :

- demander des critiques de livres ou d’articles, en créant un ticket ;
- publier des critiques de livres ou d’articles

Les fonctionnalités d'inscription et de connexion sont indispensables pour ce MVP.

Lorsqu'un utilisateur se connecte au système, son flux est la première page qu'il voit.
Il peut y voir les tickets et les avis de tous les utilisateurs qu'il suit.
Il doit également voir ses propres tickets et avis, ainsi que les avis en réponse à ses propres tickets – même si l’utilisateur qui a répondu ne fait pas partie des personnes qu’il suit.
La logique qui consiste à combiner des ensembles de requêtes de différents types de modèles peut être compliquée.
Consultez l'annexe à la fin de cette spécification pour obtenir des conseils sur la manière de procéder.

Un ticket correspond à une demande de critique de la part d'un utilisateur.
Il affiche son ticket en demandant une critique pour un livre ou un article de littérature.
Les utilisateurs qui les suivent peuvent émouvoir et poster des critiques pour des livres et des articles qui n'ont pas encore de ticket.

Un utilisateur peut suivre d'autres utilisateurs.
Comme il ne s'agit que d'un MVP (regardez des tutos sur youtube), cette fonctionnalité doit rester assez simple.
Vous n'aurez pas besoin d'une fonction de recherche ou d'un flux Discover pour les nouveaux utilisateurs, ensuite soumettre leurs critiques en réponse au ticket.

Les utilisateurs devraient également pouvoir garder une zone de texte simple, dans laquelle vous entrez le nom d'utilisateur que vous souhaitez suivre.

Vous devriez également avoir une page qui liste tous les utilisateurs que l'utilisateur connecté suit, avec l'option de dépliage qui permet de cesser de suivre un utilisateur donné.

Vous aurez également besoin d'une autre page à partir de laquelle les utilisateurs peuvent examiner leurs propres soumissions. Ils devraient pouvoir voir leurs messages, les modifier et les supprimer à partir de cette page.

N'oubliez pas qu'il s'agit d'un MVP, alors essayez de ne pas trop vous attarder sur le style. Concentrez-vous plutôt sur une interface utilisateur propre et minimale. Cependant, vous devez veiller à ce que les formats de date, le style, etc., soient cohérents sur l'ensemble du site.
Suivez la disposition des wireframes fournis, mais n'ayez pas peur d'ajouter quelques touches personnelles si vous le souhaitez – rappelez-vous, propres et minimales .

Un utilisateur devra pouvoir :

- se connecter et s'inscrire – le site ne doit pas être accessible à un utilisateur non connecté
- consulter un flux contenant les derniers tickets et les commentaires des utilisateurs qu'il suit, classés par ordre chronologique, les plus récents en premier ;
- créer de nouveaux tickets pour demander une critique sur un livre/article ;
- créer des critiques en réponse à des tickets ;
- créer des critiques qui ne sont pas en réponse à un ticket. Dans le cadre d'un processus en une étape, l'utilisateur créera un ticket puis un commentaire en réponse à son propre ticket ;
- voir, modifier et supprimer ses propres tickets et commentaires ;
- suivre les autres utilisateurs en entrant leur nom d'utilisateur ;
- voir qui il suit et suivre qui il veut ;
- cesser de suivre un utilisateur.

Un développeur devra pouvoir :

- créer un environnement local en utilisant venv, et gérer le site en se basant sur la documentation détaillée présentée dans le fichier README.md.

Le site devra :

- avoir une interface utilisateur correspondant à celle des wireframes ;
- avoir une interface utilisateur propre et minimale ;

La base de code devra :

- utiliser le framework Django ;
- utiliser SQLite comme base de données de développement locale (votre fichier db.sqlite3 doit être inclus dans le repository) ;
- avoir une conception de base de données qui correspond au schéma de la base de données ;
- avoir une syntaxe qui respecte les directives PEP 8.

Appendice

Exemple de combinaison de requêtes pour le flux à partir de différents modèles

```python
# in views.py
from itertools import chain

from django.db.models import CharField, Value
from django.shortcuts import render



def feed(request):
    reviews = get_users_viewable_reviews(request.user)
    # returns queryset of reviews
    reviews = reviews.annotate(content_type=Value('REVIEW', CharField()))

    tickets = get_users_viewable_tickets(request.user)
    # returns queryset of tickets
    tickets = tickets.annotate(content_type=Value('TICKET', CharField()))

    # combine and sort the two types of posts
    posts = sorted(
    '    chain(reviews, tickets),
        key=lambda post: post.time_created,
        reverse=True
    )
    return render(request, 'feed.html', context={'posts': posts})



# in feed.html
# Use the 'include' tag to reuse ticket and review elements between pages

...

{% for post in posts %}
    {% if post.content_type == 'TICKET' %}
        {% include 'ticket_snippet.html' %}
    {% elif post.content_type == 'REVIEW' %}
        {% include 'review_snippet.html' %}
{% endfor %}

...
```