Développement d’une application web DJANGO.

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.

Ils postent leur ticket demandant une critique pour un livre ou un article de littérature.

Les utilisateurs qui les suivent peuvent soumettre leurs critiques en réponse au ticket. Les utilisateurs doivent également pouvoir publier des critiques de livres et d'articles qui n'ont pas encore de ticket.

Les utilisateurs pourront suivre d'autres utilisateurs et devraient également avoir la possibilité de ne plus les suivre.

Comme il ne s'agit que d'un MVP, 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, mais une zone de texte simple, dans laquelle vous entrez le username de l’user que vous souhaitez suivre.

Vous devriez également avoir une page qui liste tous les utilisateurs **suivis par l'utilisateur connecté** avec l’option unfollow (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 pourront revoir leurs propres soumissions. Ils devraient pouvoir voir leurs publications, les modifier et les supprimer de cette page.

N'oubliez pas qu'il s'agit d'un MVP, alors essayez de ne pas trop vous préoccuper du style. Concentrez-vous davantage sur une interface utilisateur propre et minimale. Cependant, vous devez vous assurer que des éléments tels que les formats de date, le style, etc. sont 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, propre et minimal.**

**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 avis des utilisateurs qu'il suit, classés par ordre chronologique (heure), les plus récents en premier,
* créer de nouveaux tickets pour demander une critique sur un livre/article ;
* créer des avis en réponse à des tickets ;
* créer des avis 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 avis en réponse à son propre ticket;
* Pouvoir afficher, modifier et supprimer ses propres tickets et avis,
* suivre d’autres utilisateurs en saisissant 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 exécuter le site en se basant sur la documentation détaillée présentée dans le fichier README.md.

**Le site devra :**

* avoir une interface utilisateur (UI) correspondant à celle des wireframes ;
* avoir une interface utilisateur propre et minimale ;
* Utiliser le rendu côté serveur pour afficher dynamiquement les informations de la base de données sur la page.

**Le codebase devra :**

* utiliser le framework Django ;
* Utiliser le langage de modèle Django pour le rendu côté serveur.
* 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 conforme aux directives PEP 8.

Appendice

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

|  |
| --- |
| # 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 %}  ... |
|  |