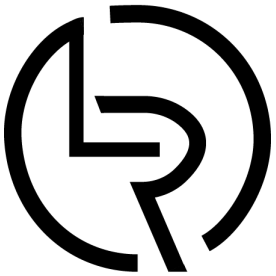
# Cahier des charges



**LITRevu**

## Résumé

L’application permet de :

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

## Les billets

Un billet correspond à une demande de critique de la part d'un utilisateur.

- Il rédige son billet en demandant une critique pour un livre ou un article de littérature.
- Les utilisateurs qui suivent l’utilisateur connecté peuvent poster des critiques en réponse à un billet de l’utilisateur connecté.
- Il est possible de créer un billet et une critique pour ce billet en même temps.

## Le flux

Lorsqu'un utilisateur se connecte au système, son flux est la première page qu'il voit. Le flux doit afficher :

- les billets et les avis de tous les utilisateurs suivis par l’utilisateur connecté ;
- les billets et avis de l’utilisateur connecté ;

- les avis en réponse aux billets de l'utilisateur connecté – même si l'utilisateur qui a répondu ne fait pas partie des personnes suivies par l'utilisateur connecté ;
- l'option de créer un billet et une critique sur ce même billet en une seule étape – c'est une méthode pour créer une critique à partir de zéro, c'est-à-dire pas nécessairement en réponse à un billet d'un autre utilisateur.

Le flux doit être ordonné par ordre antéchronologique (les plus récents en premier), selon la dernière action effectuée.

***Consultez l'appendice à la fin de cette spécification pour obtenir des conseils sur la combinaison des ensembles de requêtes de différents types de modèles.***

## Les utilisateurs suivis

Un utilisateur peut suivre d'autres utilisateurs pour voir leurs critiques (ce qu'ils demandent et ce qu'ils postent). Il n'est pas nécessaire de créer un système complexe de suivi pour ce projet (flux Discover ou recherche d'utilisateurs).

Les utilisateurs devraient avoir un champ texte simple dans lequel ils entrent le nom de l'utilisateur qu'ils souhaitent suivre. Si un utilisateur tape le nom d'une personne qui n'existe pas, il faudra prévoir un message pour lui notifier.

Ils devraient également avoir accès à une page listant tous les utilisateurs suivis avec une option pour ne plus suivre un utilisateur en particulier.

## L'authentification

Le site doit proposer une page d'inscription et une page de connexion.

Un utilisateur non connecté ne devrait avoir accès qu'aux pages d'inscription et de connexion.

## Les fonctionnalités

Un visiteur non connecté doit pouvoir :

- s'inscrire ;
- se connecter.

Un utilisateur connecté doit pouvoir :

- consulter son flux contenant les derniers billets et les critiques des utilisateurs qu'il suit, classés par ordre antichronologique ;

- créer des nouveaux billets pour demander des critiques sur un livre/un article ;
- créer des nouvelles critiques en réponse à des billets ;
- créer un billet et une critique sur ce même billet en une seule étape, pour créer des critiques « à partir de zéro »
- voir, modifier et supprimer ses propres billets et critiques ;
- suivre les autres utilisateurs en entrant leur nom d'utilisateur ;
- voir qui il suit et suivre qui il veut ;
- arrêter de suivre ou bloquer un utilisateur.

Un développeur devra pouvoir :

- créer un environnement local 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 dans son architecture, le design restant assez libre ;
- avoir une interface utilisateur propre et minimale.

## Les spécifications techniques

- Utiliser le framework Django.
- Utiliser SQLite comme base de données locale (le fichier db.sqlite3 doit être inclus dans le repository).
- Offrir une conception de base de données correspondant au schéma fourni.
- Respecter les directives de la PEP8.

## 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 %}
...
```