# Requirements Specification

## Summary

Signup and login functionality are a must in terms of what you need for this MVP. When a user logs into the system, their feed is the first page they will see. There they can see the tickets and reviews of all users they follow. They should also see their own tickets and reviews, as well as any reviews in response to their own tickets - even if they do not follow the reviewer. (The logic around combining querysets of different model types can be complicated. Check the appendix at the end of this specification for some guidance on how to do this.)

You can think of a ticket as a request for a review from a user. They post their ticket requesting a review for a book or literature article. Users who follow them can then submit their reviews in response to the ticket. Users should also be able to post reviews for books and articles that do not have a ticket yet.

Users will be able to follow other users and should also have the option to unfollow them. As this is just an MVP, keep this functionality fairly simple. You won’t need a search functionality or a Discover feed for new users. Keep it to a simple box where you enter the username of the user you wish to follow. You should also have a page that lists all the users the logged-in user follows with the option to unfollow.

You will also need another page from which users can review their own submissions. They should be able to see their posts and edit and delete them from this page.

Remember, this is an MVP, so try not to get too caught up on styling. Focus more on a clean and minimal UI. However, you should ensure things like date formats, styling, etc. are consistent across the site. Follow the layout of the wireframes provided, but don’t be afraid to add some personal touches if you wish, remember, clean and minimal.

**A user will need to:**

* Log in and sign up - the site should not be accessible to a non-logged-in user.
* View a feed containing the latest tickets and reviews from users that they follow ordered by time with the latest first.
* Create new tickets requesting a review on a book/article.
* Create reviews as a response to tickets.
* Create reviews not in response to a ticket. As part of a one-step process, the user will create a ticket and then a review responding to their own ticket.
* Be able to view, edit, and delete their own tickets and reviews,
* Follow other users by entering their username,
* View who they follow and unfollow whoever they want.

**A developer will need to:**

* Create a local environment using venv and run the site based on the detailed documentation laid out in the README.md.

**The site will need to:**

* Have a UI matching those of the wireframes.
* Have a clean and minimal UI.
* Use server-side rendering to display information from the database on the page dynamically.

**The codebase will need to:**

* Use the Django framework.
* Use the Django template language for server-side rendering.
* Use SQLite as a local development DB (your db.sqlite3 file should be included in the repository).
* Have a database design that matches the database schema. Have syntax that meets PEP8 guidelines.

## 

## Appendix

### Example for combining querysets of different models for the Feed

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

# **(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**

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