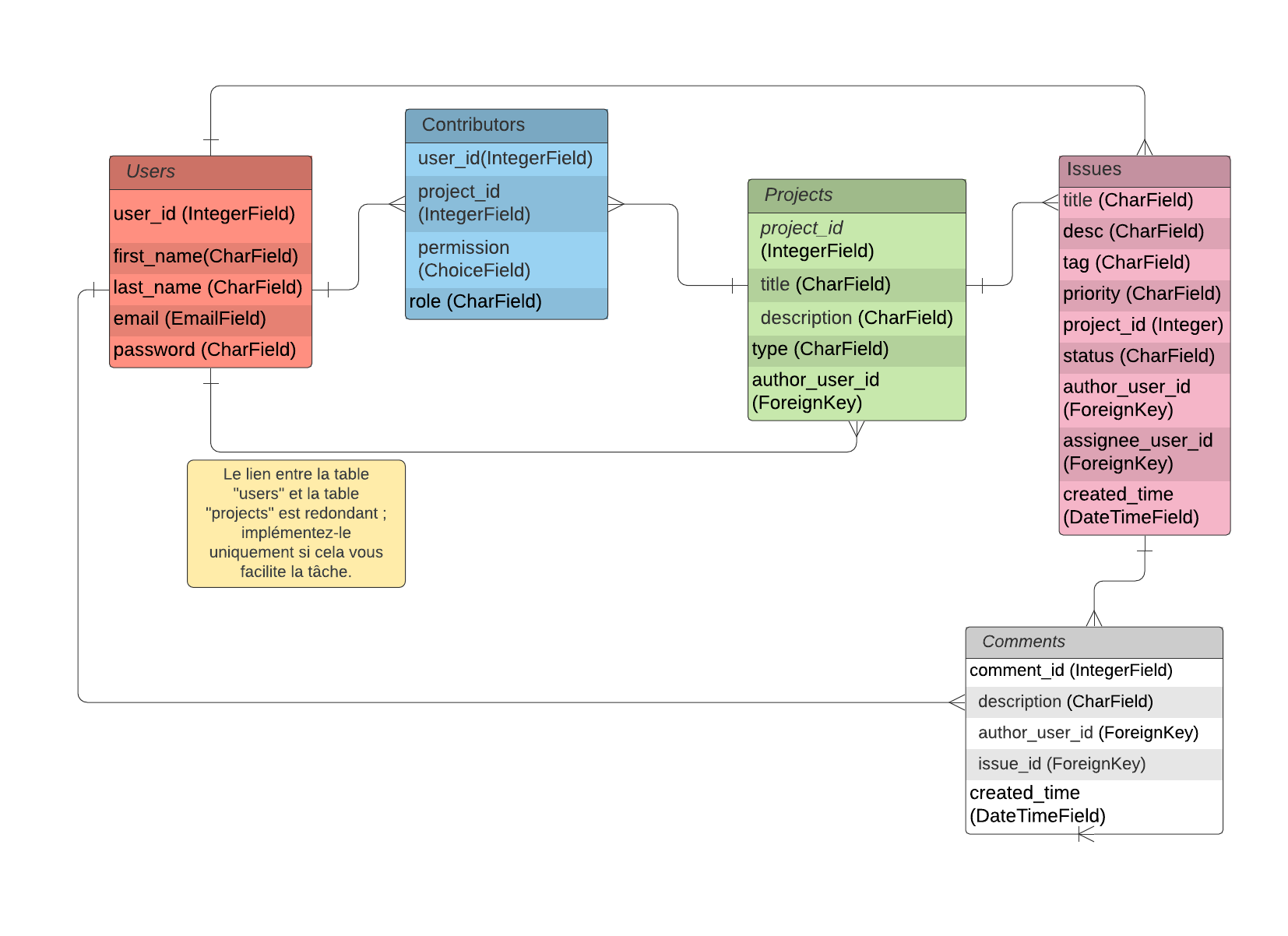
Conception de la mise en œuvre

Ce document fournit tous les détails nécessaires pour vous aider à comprendre la logique métier de l'API REST que nous voulons développer, et notamment :

1. Un diagramme de classes pour identifier le modèle de base de données. (veillez à identifier correctement les modèles et les relations) ;
2. La définition de chaque table dans le diagramme de classes ;
3. La liste de tous les points de terminaison d'API.

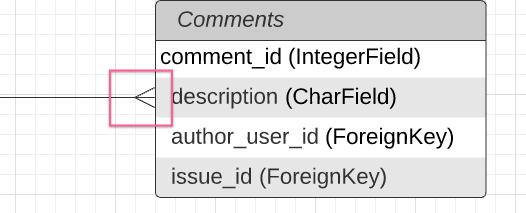
## Diagramme de classes du système de suivi des problèmes



*Vous pouvez télécharger l’image via* [*ce lien*](https://s3-eu-west-1.amazonaws.com/course.oc-static.com/projects/Python+FR/P8+-+Cr%C3%A9ez+une+API+s%C3%A9curis%C3%A9e+RESTful+en+utilisant+Django+REST+/Class+Diagram+for+P8.png)*.*

Chaque table représente une entité (un modèle) dans le projet. Les champs d'attributs dans chaque table doivent être répercutés dans votre API (remarque : vous pouvez ajouter d'autres champs si vous le souhaitez). Le type des champs est indiqué dans un modèle Django. Chaque champ peut également être un sérialiseur, conformément à votre mise en œuvre.

## Définition des tables :

1. **Users** : elle stocke les identifiants de connexion des utilisateurs.
2. **Projects** : entité qui stocke toutes les informations concernant chaque projet/produit/application en cours de développement ou de gestion dans l'entreprise. Elle entretient une relation plusieurs-à-plusieurs avec la table des utilisateurs, via une table de jonction appelée Contributors.  
   Par souci de simplicité, nous avons également ajouté une relation un-à-plusieurs avec la table **Users**, de sorte que nous pouvons enregistrer **l'auteur/le responsable/le créateur** du projet. Vous pouvez également gérer ce cas à l'aide du champ d'autorisation de la classe Contributor.
3. **Contributor :** table *through* permettant d'établir la relation plusieurs-à-plusieurs entre la table Users et la table Projects.
4. **Issues :** elle stocke tous les problèmes d'un projet, ainsi que leurs statut, priorité, attributaire (utilisateur auquel le problème est affecté), balise (bug, tâche, amélioration), et d'autres détails nécessaires mentionnés dans la table. Elle a une relation plusieurs-à-un avec la table **Projects**, et une autre relation plusieurs-à-un avec la table **Users**.
5. **Comments :** stocke tous les commentaires d'un problème particulier, et a donc une relation plusieurs-à-un avec la table **Issues**. Chaque commentaire contient des détails tels que description, user\_id (relation plusieurs-à-un avec la table **Users**).
6. **Relations :**

La zone de l'image mise en évidence montre une relation un-à-plusieurs, et indique la nécessité d'un champ ForeignKey. Par exemple, le côté ramifié ici montre que de nombreuses instances de la table Comments peuvent appartenir à une même instance de la table Users.

## Liste des points de terminaison d'API requis et de leurs URI

Pour créer les routages (URL) pour chaque fonctionnalité ci-dessous:

Chaque point de terminaison doit renvoyer une réponse de réussite 200 avec les **données JSON** fournissant tous les détails du sérialiseur.

Un code de statut approprié doit être renvoyé en cas d'erreur.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Point de terminaison d'API | Méthode HTTP | URI |
| 1. | Inscription de l'utilisateur | POST | /signup/  RG: ouvert à tout utilisateur |
| 2. | Connexion de l'utilisateur | POST | /login/  RG: ouvert à tout utilisateur |
| 3. | Récupérer la liste de tous les projets (projects) rattachés à l'utilisateur (user) connecté | GET | /projects/  RG: ouvert à tout utilisateur authentifié, ne remonte que les projets dont il est contributeur. |
| 4. | Créer un projet | POST | /projects/  RG: possible pour n’importe quel utilisateur connecté. Créé automatiquement l’utilisateur en tant que contributeur / auteur. |
| 5. | Récupérer les détails d'un projet (project) via son id | GET | /projects/{id}/  RG: ouvert à tout utilisateur authentifié, info ne remontant que si l’user est contributeur. |
| 6. | Mettre à jour un projet | PUT | /projects/{id}/  RG: possible que pour l’auteur du projet |
| 7. | Supprimer un projet et ses problèmes | DELETE | /projects/{id}/  A CHECKER + EXEMPLE |
| 8. | Ajouter un utilisateur (collaborateur) à un projet | POST | /projects/{id}/users/  RG: ouvert à tout collaborateur du projet en question. |
| 9. | Récupérer la liste de tous les utilisateurs (users) attachés à un projet (project) | GET | /projects/{id}/users/  RG: ouvert seulement aux collaborateurs du projet |
| 10. | Supprimer un utilisateur d'un projet | DELETE | /projects/{id}/users/{id}  RG: ouvert seulement à l’auteur du projet. Ne supprime pas issues et comments du contributeur supprimés. |
| 11. | Récupérer la liste des problèmes (issues) liés à un projet (project) | GET | /projects/{id}/issues/  RG: ouvert seulement aux collaborateurs du projet |
| 12. | Créer un problème dans un projet | POST | /projects/{id}/issues/  RG: ouvert seulement aux collaborateurs du projet |
| 13. | Mettre à jour un problème dans un projet | PUT | /projects/{id}/issues/{id}  RG: seul le collaborateur ayant créé le problème (author\_user\_id) |
| 14. | Supprimer un problème d'un projet | DELETE | /projects/{id}/issues/{id}  RG: seul le collaborateur ayant créé le problème (author\_user\_id)  Et ses commentaires ? |
| 15. | Créer des commentaires sur un problème | POST | /projects/{id}/issues/{id}/comments/  RG: ouvert à tout collaborateur du projet en question. |
| 16. | Récupérer la liste de tous les commentaires liés à un problème (issue) | GET | /projects/{id}/issues/{id}/comments/  RG: ouvert à tout collaborateur du projet en question. |
| 17. | Modifier un commentaire | PUT | /projects/{id}/issues/{id}/comments/{id}  RG: seulement pour l’auteur du commentaire en question |
| 18. | Supprimer un commentaire | DELETE | /projects/{id}/issues/{id}/comments/{id}  RG: seulement pour l’auteur du commentaire en question |
| 19. | Récupérer un commentaire (comment) via son id | GET | /projects/{id}/issues/{id}/comments/{id}  RG: ouvert à tout collaborateur du projet en question. |

## Documentation des points de terminaison d'API

Tous les points de terminaison peuvent être testés à l'aide de l’application Postman (recommandé) ou de n'importe quel autre outil, comme la commande “curl” ou le serveur localhost du Django REST Framework.

Les points de terminaison d'API doivent être documentés sur Postman. Vous devrez partager la documentation en la publiant sur le Web.

### Étapes pour documenter votre API

1. Créez une [collection Postman](https://learning.postman.com/docs/postman/collections/intro-to-collections/) de tous les points de terminaison (requests).

2. Suivez ce [guide de documentation de Postman](https://learning.postman.com/docs/postman/api-documentation/documenting-your-api/) pour documenter une collection de points de terminaison existante, en donnant à chacun d'eux une courte description.

3. Partagez la documentation en la publiant sur le Web, comme indiqué [ici](https://learning.postman.com/docs/postman/api-documentation/publishing-your-docs/).