

SoftDesk Support

API RESTful, support technique et sécurités

1. Modèles, tables relationnelles

2. Construction de l'API

3. Opérations CRUD et endpoints

4. Tests unitaires

5. Sécurités

4. Green Code



SoftDesk
Support

1. Modèles

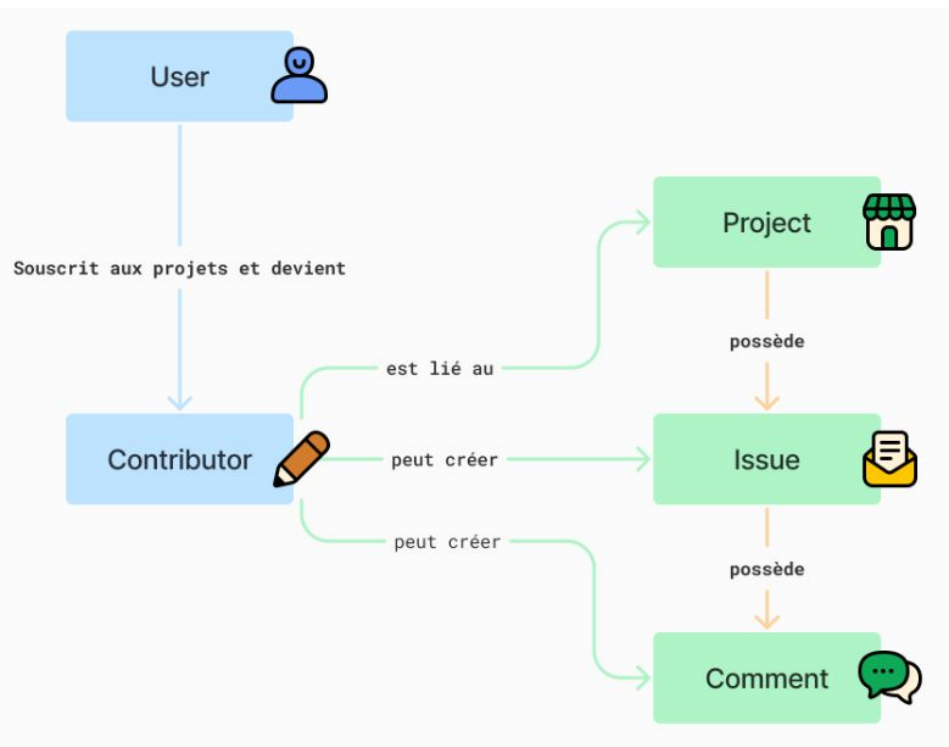



Diagramme des relations
du système de suivi des
problèmes.

Table Modèles

Table +

≡ ↑↓ ⚡ 🔍 ...

Nouveau ▾

Aa Nom	Description	Classe hérité	Details	+ ...
 <u>User</u>	Définit les utilisateurs	AbstractUser		
 <u>Contributor</u>	Définit les utilisateurs qui sont contributeurs d'un projet spécifique... Un utilisateur peut contribuer à plusieurs projets, et un projet peut avoir plusieurs contributeurs. Le contributeur peut créer trois types de ressources: le project, l'issue et le comment.	models.Model	Classe Intermédiaire (User, Project)	
 <u>Project</u>	Définit les projets d'une application cliente. C'est la ressource principale utilisée par le client.	models.Model		
 <u>Issue</u>	Définit les problèmes d'un projet, ainsi que son statut, sa priorité, son attribution...	models.Model		
 <u>Comment</u>	Définit les commentaires d'un problème (issue) particulier.	models.Model		

+ Nouveau



Project

- Description Définit les projets d'une application cliente. C'est la ressource principale utilisée par le client.
- Classe hérité `models.Model`
- Details Vide
- Ajouter une propriété

 Ajouter un commentaire...

✓ 2 liens entrants

Table

Aa Champs	Descriptions	FieldType	Options/Values	Table Modèle
name	Le nom du projet	Charfield		
description	La description du projet	TextField		
type	Le type de projet	Charfield	CHOICES=Back-end, Front-end, IOS, Android	
author	L'user qui créer le projet, il devient Contributor	ForeignKey		 User
contributors	Les contributors qui peuvent être assignés au projet	ForeignKey		 Contributor
created_time	Date de création	DateTimeField	auto-now-add=True	

+ Nouveau

Une organisation maîtrisée nous permet :

- Une vue d'ensemble et une représentation efficace
- Une réduction des risques d'incohérences
- Une communication efficace en équipe

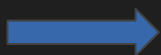
2. Construction de l'API

- Django REST_Framework
- Utilisation des modelViewSet et Routeurs
 - Les différentes Urls
 - Mixin pour les serializers
- Les Serializers
 - Validations personnalisés
 - Surcharge de méthodes
- La construction des views



SoftDesk
Support

```
router = routers.SimpleRouter()
```



Permet la création dynamique des routes

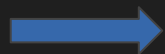
```
router.register(r'users', UserViewSet, basename='user')
router.register(r'contributors', ContributorViewSet, basename='contributor')
router.register(r'projects', ProjectViewSet, basename='project')
router.register(r'projects/(?P<project_pk>\d+)/issues', IssueViewSet, basename='issue')
router.register(r'projects/(?P<project_pk>\d+)/issues/(?P<issue_pk>\d+)/comments', CommentViewSet, basename='comment')
```

```
urlpatterns = [
```

```
    # API & Admin
    path('admin/', admin.site.urls),
    path('api-auth/', include('rest_framework.urls')),
    path('api/', include(router.urls)),
```

```
    # Token
```

```
    path('api/login/', TokenObtainPairView.as_view(), name='login'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
```



Obtention des token

```
class UserListSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=False)
    password_confirm = serializers.CharField(write_only=True, required=False)

    class Meta:
        model = User
        fields = ['id', 'username', 'password', 'password_confirm', 'email', 'age', 'can_be_contacted', 'can_data_be_shared']
```

```
def validate(self, data):
    password = data.get('password')
    password_confirm = data.get('password_confirm')
    if password or password_confirm:
        if password != password_confirm:
            raise serializers.ValidationError("Les mots de passe ne correspondent pas.")
    return data

def validate_age(self, value):
    if value < 15:
        raise serializers.ValidationError('Vous devez avoir au moins 15ans')
    return value

def create(self, validated_data):
    # Remove password_confirm from the validated data
    validated_data.pop('password_confirm')
    # Create a new user with the validated data
    user = User.objects.create_user(**validated_data)
    return user
```




Validations
personnalisées
pour les données
entrantes


```
class MultipleSerializerMixin:
```

```
    detail_serializer_class = None
```

```
    def get_serializer_class(self):  
        if self.action in ['retrieve', 'update', 'partial_update', 'destroy'] and self.detail_serializer_class is not None:  
            return self.detail_serializer_class  
        return super().get_serializer_class()
```


Choisis dynamiquement le
serializer



```
class ProjectViewSet(MultipleSerializerMixin, ModelViewSet):  
    serializer_class = ProjectListSerializer  
    detail_serializer_class = ProjectDetailSerializer
```

```
    def get_permissions(self):  
        match self.action:  
            case 'list' | 'create':  
                self.permission_classes = [IsAuthenticated]  
            case 'retrieve':  
                self.permission_classes = [IsAuthenticated, IsProjectContributor]  
            case _:  
                self.permission_classes = [IsAuthenticated, IsProjectContributor, IsAuthor]  
        return super().get_permissions()
```

Sélection de la
permission en
fonction de l'action



```
    def get_queryset(self):  
        return Project.objects.all()
```

4. Les Tests Unitaires

Avantages et importances

Environnement de tests

Efficacité de développement et sérénité

Intégration en continue

5. Sécurités

Importance de la sécurité dans les API

JSON Web Token

Dépendabot

Permission personnalisés

5. Philosophie “Green Code”

Comprendre l'impact

Transfert des données

Pagination

CONCLUSION

- Suivis minutieux du cahier des charges
- Sécurités renforcée
- Optimisation et performance
- Intégration continue et qualité de code
- Engagement en développement durable