**Projet : Application de Gestion de Ressources Communautaires**

Cette application permettrait à des utilisateurs de gérer des ressources (ex : véhicules partagés, salles de réunion, outils communs, etc.) pour une communauté comme une résidence, une école ou un espace de coworking. Les utilisateurs peuvent réserver, emprunter, rendre et suivre la disponibilité des ressources.

**Fonctionnalités principales**

1. **Gestion des Utilisateurs et des Rôles**
   * Les utilisateurs peuvent avoir différents rôles (administrateur, membre, visiteur).
   * Chaque rôle a des permissions spécifiques : création de ressources, réservation, accès limité, etc.
2. **Système de Réservation**
   * Les utilisateurs peuvent voir les ressources disponibles et les réserver pour une date et une période spécifiques.
   * Les administrateurs peuvent approuver ou rejeter les réservations.
3. **Gestion des Ressources**
   * Permet de créer, modifier et supprimer des ressources (ex : ajouter un véhicule de covoiturage ou une salle de réunion).
   * Suivi de l'état de la ressource (ex : disponible, en maintenance, en cours d'utilisation).
4. **Suivi et Historique des Emprunts**
   * Enregistrer l’historique des réservations, incluant les utilisateurs et les périodes d’utilisation.
   * Générer des rapports pour visualiser l'utilisation des ressources au fil du temps.
5. **Notifications et Alertes**
   * Alerter les utilisateurs lorsque leur réservation approche ou lorsqu’une ressource est de nouveau disponible.
   * Notifications pour les administrateurs lorsque des ressources nécessitent une attention (maintenance, nettoyage).
6. **Gestion des Incidents**
   * Signalement des problèmes par les utilisateurs (ex : un véhicule qui nécessite des réparations).
   * Suivi des actions prises pour résoudre les problèmes signalés.
7. **Système de Recherche et Filtrage**
   * Permettre aux utilisateurs de rechercher et de filtrer les ressources par type, disponibilité, etc.

**Organisation des tâches pour un projet collaboratif**

**1. Conception et Planification**

* **Conception des modèles de données** : Définir les entités (utilisateur, ressource, réservation, historique, incident).
* **Wireframes et UX** : Créer des croquis pour l'interface utilisateur pour la réservation, la gestion des ressources, etc.

**2. Back-end (Django)**

* **Gestion des utilisateurs** : Modèle utilisateur, rôles et permissions.
* **Création des modèles de ressource** : Modèle pour les ressources partagées avec attributs.
* **Système de réservation** : Modèle et logique de gestion des réservations.
* **Suivi et historique** : Modèle pour l’historique des emprunts et réservations.
* **Gestion des notifications** : Création des alertes et notifications (email ou système interne).
* **Signalement des incidents** : Logique pour le signalement et suivi des incidents.

**3. Front-end**

* **Page d'accueil et tableau de bord** : Vue d’ensemble des ressources disponibles et des réservations.
* **Page de gestion des ressources** : Interface pour lister et ajouter des ressources.
* **Page de réservation** : Interface pour réserver une ressource et vérifier la disponibilité.
* **Page d'incidents** : Interface pour signaler et suivre les incidents.

**4. Tests et Documentation**

* **Tests unitaires** : Tests pour le modèle de réservation, la gestion des ressources, les incidents, etc.
* **Documentation** : Explications pour l'installation, l’utilisation des API, et guides pour les contributeurs.

**5. Déploiement et Maintenance**

* **Préparation pour le déploiement** : Configuration pour une plateforme comme Heroku ou AWS.
* **Déploiement continu et maintenance** : Suivi et amélioration continue avec des pipelines CI/CD.

from django.db import models

from django.contrib.auth.models import AbstractUser

# Modèle Utilisateur avec rôles (hérite d'AbstractUser pour la personnalisation)

class User(AbstractUser):

    ROLES = [

        ('admin', 'Administrateur'),

        ('member', 'Membre'),

        ('visitor', 'Visiteur'),

    ]

    role = models.CharField(max\_length=10, choices=ROLES, default='visitor')

    def \_\_str\_\_(self):

        return f"{self.username} ({self.get\_role\_display()})"

# Modèle Ressource

class Resource(models.Model):

    RESOURCE\_TYPES = [

        ('vehicle', 'Véhicule'),

        ('room', 'Salle de réunion'),

        ('tool', 'Outil'),

    ]

    type = models.CharField(max\_length=20, choices=RESOURCE\_TYPES)

    name = models.CharField(max\_length=100)

    description = models.TextField(blank=True, null=True)

    status = models.CharField(

        max\_length=20,

        choices=[('available', 'Disponible'), ('in\_use', 'En cours d\'utilisation'), ('maintenance', 'En maintenance')],

        default='available'

    )

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return self.name

# Modèle Réservation (avec relation Many-to-One vers Ressource et Utilisateur)

class Reservation(models.Model):

is\_paid = models.BooleanField(default=False)  # Indique si la réservation a été payée

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='reservations')

    user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='reservations')

    start\_date = models.DateTimeField()

    end\_date = models.DateTimeField()

    status = models.CharField(

        max\_length=20,

        choices=[('pending', 'En attente'), ('approved', 'Approuvé'), ('rejected', 'Rejeté')],

        default='pending'

    )

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return f"Reservation by {self.user} for {self.resource}"

# Modèle Historique d’Emprunt

class UsageHistory(models.Model):

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='usage\_history')

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    reservation = models.ForeignKey(Reservation, on\_delete=models.CASCADE)

    usage\_date = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"{self.user} used {self.resource} on {self.usage\_date}"

# Modèle Notification

class Notification(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='notifications')

    message = models.TextField()

    is\_read = models.BooleanField(default=False)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"Notification for {self.user}: {self.message}"

# Modèle Incident (pour les problèmes signalés par les utilisateurs)

class Incident(models.Model):

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='incidents')

    reported\_by = models.ForeignKey(User, on\_delete=models.CASCADE)

    description = models.TextField()

    status = models.CharField(

        max\_length=20,

        choices=[('reported', 'Signalé'), ('in\_progress', 'En cours'), ('resolved', 'Résolu')],

        default='reported'

    )

    created\_at = models.DateTimeField(auto\_now\_add=True)

    updated\_at = models.DateTimeField(auto\_now=True)

    def \_\_str\_\_(self):

        return f"Incident on {self.resource} by {self.reported\_by}"

**Explication des Relations**

1. **User** : Utilisateur avec un rôle spécifique.
2. **Resource** : Représente une ressource communautaire (par exemple, salle, véhicule).
3. **Reservation** : Associe un utilisateur à une ressource pour une réservation spécifique (relation avec User et Resource).
4. **UsageHistory** : Enregistre l'utilisation d'une ressource pour créer des historiques d'emprunts.
5. **Notification** : Gère les notifications pour les utilisateurs.
6. **Incident** : Enregistre les incidents signalés et leurs statuts pour chaque ressource.

**Modèles supplémentaires**

1. **Feedback (Retour d'expérience)**
   * Permettrait aux utilisateurs de donner leur avis sur une ressource après l’avoir utilisée.
   * **Relations** : Liaison avec User et Resource.

class Feedback(models.Model):

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='feedback')

    user = models.ForeignKey(User, on\_delete=models.CASCADE)

    rating = models.IntegerField(choices=[(i, i) for i in range(1, 6)])  # Évaluation de 1 à 5

    comment = models.TextField(blank=True, null=True)

    created\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"Feedback by {self.user} for {self.resource}"

**Maintenance Log (Journal de Maintenance)**

* Gère l'historique de maintenance d'une ressource pour suivre les réparations, inspections, etc.
* **Relations** : Liaison avec Resource.

class MaintenanceLog(models.Model):

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='maintenance\_logs')

    description = models.TextField()

    performed\_by = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True)  # Technicien ou admin

    date = models.DateField()

    is\_resolved = models.BooleanField(default=False)

    def \_\_str\_\_(self):

        return f"Maintenance for {self.resource} on {self.date}"

**Payment (Paiement)**

* Si l’utilisation de certaines ressources est payante, on peut gérer les paiements des utilisateurs avec un modèle de transactions.
* **Relations** : Liaison avec User et Reservation.

class Payment(models.Model):

    user = models.ForeignKey(User, on\_delete=models.CASCADE, related\_name='payments')

    reservation = models.ForeignKey(Reservation, on\_delete=models.SET\_NULL, null=True, blank=True)

    amount = models.DecimalField(max\_digits=10, decimal\_places=2)

    date = models.DateTimeField(auto\_now\_add=True)

    status = models.CharField(max\_length=20, choices=[('pending', 'En attente'), ('completed', 'Complété'), ('failed', 'Échoué')])

    def \_\_str\_\_(self):

        return f"Payment of {self.amount} by {self.user}"

**Document (Document associé aux ressources)**

* Pour chaque ressource, il pourrait y avoir des documents associés (ex : manuel d'utilisation, certificats, etc.).
* **Relations** : Liaison avec Resource.

class Document(models.Model):

    resource = models.ForeignKey(Resource, on\_delete=models.CASCADE, related\_name='documents')

    title = models.CharField(max\_length=100)

    file = models.FileField(upload\_to='documents/')

    uploaded\_at = models.DateTimeField(auto\_now\_add=True)

    def \_\_str\_\_(self):

        return f"Document {self.title} for {self.resource}"

**Explication des modifications**

* **Reservation.is\_paid** : Champ pour indiquer si la réservation a été payée.
* **Payment** : Associe chaque paiement à une réservation. Lorsque le paiement est **completed**, Reservation.is\_paid peut être mis à jour pour faciliter le suivi des emprunts payants.