

Le document fourni décrit en détail les responsabilités du backend dans le cadre du projet de développement d'une application éducative interactive eCEP. Voici un résumé des tâches principales que le backend doit accomplir :

1. Gestion des utilisateurs et authentification

- Création et gestion des comptes utilisateurs (élèves, enseignants, parents, administrateurs).
- Authentification sécurisée avec **JWT (JSON Web Tokens)**.
- Gestion des rôles et permissions (élèves, enseignants, parents, administrateurs).
- Système de récupération de mot de passe (exemple : réinitialisation par email).

2. Gestion des contenus pédagogiques

- Création, mise à jour et suppression des contenus (cours multimédias, exercices interactifs, examens types).
- Organisation des contenus par matière, chapitre et niveau scolaire.
- Gestion des fichiers multimédias (stockage, récupération, mise à jour).
- Validation des contenus par les administrateurs avant publication.

3. Système d'évaluation et de suivi

- Génération d'examens types (examens blancs ou annuels).
- Correction des évaluations et attribution de notes.
- Suivi des résultats des élèves (historique des performances, statistiques).
- Génération de rapports détaillés pour les enseignants et parents.
- Enregistrement des exercices et badges obtenus.

4. Gestion des badges, défis et récompenses (gamification)

- Définition des règles pour l'attribution de badges, défis et récompenses.
- Suivi des progrès des élèves et mise à jour des récompenses.
- Gestion des défis (exemple : terminer un chapitre en un temps donné).

5. Mode hors ligne et synchronisation

- Génération de fichiers ou datasets pour le téléchargement des cours et exercices.
- Synchronisation des données entre le mode hors ligne et en ligne (enregistrement des progrès hors ligne, mise à jour automatique des données sur le serveur).

6. Recherche et navigation dans les contenus

- Système de recherche par mots-clés (matières, chapitres, titres des cours).
- Filtres avancés (par matière, chapitre, type de contenu).
- Pagination pour afficher les résultats de manière optimisée.

7. Module de gestion des examens

- Création, mise à jour et suppression des sessions d'examens.
- Gestion des inscriptions aux examens (élèves).
- Correction automatique des examens pour les exercices à choix multiples (QCM).
- Génération de rapports d'examens (notes, statistiques, erreurs fréquentes).

8. Tableau de bord

- Résumé des apprentissages pour les élèves (cours suivis, exercices réussis, badges obtenus).
- Suivi des performances des élèves pour les enseignants.
- Suivi des performances des enfants pour les parents.
- Gestion globale des utilisateurs et contenus pour les administrateurs.

9. Gestion des licences et paiements

- Génération de numéros de série uniques pour chaque licence.
- Intégration d'un système de paiement sécurisé pour l'activation des licences.
- Suivi des licences actives et expirées.

10. Notifications et alertes

- Gestion des notifications push (rappels d'examens, nouveaux cours, récompenses obtenues).
- Gestion des préférences de notification par utilisateur.

11. Sécurité et performances

- Sécurisation des API avec **JWT** pour l'authentification.
- Chiffrement des mots de passe et données sensibles.
- Protection contre les attaques courantes (SQL Injection, CSRF, etc.).
- Optimisation des requêtes pour de meilleures performances.

12. Hébergement et gestion des fichiers

- Hébergement des fichiers multimédias (vidéos, images, audios).
- Gestion des accès aux fichiers (téléchargement sécurisé).
- Préparation pour le déploiement sur une plateforme d'hébergement (configuration du serveur, déploiement).

Technologies utilisées

- **Backend** : Django (Python) pour la gestion des données et la logique métier.
- **Base de données** : PostgreSQL pour le stockage des données.
- **Authentification** : Django REST Framework avec JWT (JSON Web Tokens).
- **Stockage hors ligne** : IndexedDB / RxDB pour la version web, SQLite pour les versions mobiles.

Résumé des responsabilités du backend

Le backend est responsable de la logique métier, de la gestion des données, de la sécurité, et des interactions avec le frontend. Il doit fournir des API via le **Django REST Framework (DRF)** pour permettre la communication avec les applications frontend (Flutter pour mobile et web).

En résumé, le backend joue un rôle central dans la gestion des utilisateurs, des contenus pédagogiques, des évaluations, de la gamification, des notifications, et de la sécurité, tout en assurant une expérience utilisateur fluide et sécurisée.

commence avec le 1

1. Gestion des utilisateurs et authentification

Le backend doit gérer la création, l'authentification et la gestion des comptes utilisateurs pour les différents acteurs de l'application : élèves, enseignants, parents et administrateurs. Voici les tâches spécifiques à accomplir :

1.1 Création et gestion des comptes utilisateurs

- **Fonctionnalités :**
 - **Inscription des utilisateurs** : Permettre aux élèves, enseignants, parents et administrateurs de créer un compte en fournissant les informations nécessaires (nom, prénom, email, mot de passe, etc.).
 - **Validation des comptes** : Mettre en place un système de validation des comptes (par exemple, via un email de confirmation).
 - **Gestion des profils** : Permettre aux utilisateurs de mettre à jour leurs informations personnelles (mot de passe, email, etc.).
 - **Suppression des comptes** : Offrir la possibilité de supprimer un compte utilisateur (pour les administrateurs ou les utilisateurs eux-mêmes).

1.2 Authentification sécurisée avec JWT (JSON Web Tokens)

- **Fonctionnalités :**
 - **Connexion des utilisateurs** : Mettre en place un système de connexion sécurisé où les utilisateurs peuvent s'authentifier avec leur email et mot de passe.
 - **Génération de tokens JWT** : Après une connexion réussie, le backend doit générer un token JWT qui sera utilisé pour authentifier les requêtes ultérieures de l'utilisateur.
 - **Validation des tokens** : Le backend doit valider les tokens JWT à chaque requête pour s'assurer que l'utilisateur est bien authentifié.
 - **Expiration des tokens** : Les tokens JWT doivent avoir une durée de vie limitée (par exemple, 24 heures) pour des raisons de sécurité.
 - **Renouvellement des tokens** : Mettre en place un mécanisme pour renouveler les tokens expirés sans obliger l'utilisateur à se reconnecter.

1.3 Gestion des rôles et permissions

- **Fonctionnalités :**

- **Attribution des rôles :** Assigner des rôles spécifiques aux utilisateurs (élève, enseignant, parent, administrateur) lors de la création de leur compte.
- **Gestion des permissions :** Définir les permissions associées à chaque rôle :
 - **Élèves :** Accès aux cours, exercices, récompenses, badges, etc.
 - **Enseignants :** Accès au tableau de bord pour le suivi des élèves, gestion des évaluations, création de contenus pédagogiques.
 - **Parents :** Accès au suivi des performances de leur enfant, recommandations personnalisées.
 - **Administrateurs :** Gestion globale des utilisateurs, modération des contenus pédagogiques, validation des ressources.
- **Vérification des permissions :** Le backend doit vérifier les permissions de l'utilisateur avant d'autoriser l'accès à certaines fonctionnalités ou ressources.

1.4 Système de récupération de mot de passe

- **Fonctionnalités :**

- **Demande de réinitialisation :** Permettre aux utilisateurs de demander une réinitialisation de mot de passe en fournissant leur email.
- **Envoi d'un email de réinitialisation :** Envoyer un email contenant un lien sécurisé pour réinitialiser le mot de passe.
- **Réinitialisation du mot de passe :** Permettre à l'utilisateur de définir un nouveau mot de passe après avoir cliqué sur le lien sécurisé.
- **Validation du lien de réinitialisation :** Le backend doit valider que le lien de réinitialisation est valide et n'a pas expiré.

1.5 Sécurité des données utilisateurs

- **Fonctionnalités :**

- **Chiffrement des mots de passe :** Les mots de passe des utilisateurs doivent être stockés de manière sécurisée (hachés et salés) dans la base de données.
- **Protection contre les attaques :** Mettre en place des mesures de sécurité pour protéger les comptes utilisateurs contre les attaques courantes (brute force, credential stuffing, etc.).
- **Journalisation des activités :** Enregistrer les activités sensibles (connexions, modifications de mot de passe, etc.) pour un suivi en cas de problème de sécurité.

1.6 API pour la gestion des utilisateurs

- **Fonctionnalités :**

- **Endpoints pour l'inscription :** Créer des endpoints API pour permettre l'inscription des utilisateurs.
- **Endpoints pour la connexion :** Créer des endpoints API pour permettre la connexion des utilisateurs et la génération de tokens JWT.
- **Endpoints pour la gestion des profils :** Créer des endpoints API pour permettre aux utilisateurs de mettre à jour leurs informations personnelles.
- **Endpoints pour la récupération de mot de passe :** Créer des endpoints API pour gérer la récupération et la réinitialisation des mots de passe.

1.7 Tests et validation

- **Fonctionnalités :**
 - **Tests unitaires** : Écrire des tests unitaires pour vérifier que chaque fonctionnalité de gestion des utilisateurs fonctionne correctement.
 - **Tests d'intégration** : Tester l'intégration entre les différentes parties du système (authentification, gestion des rôles, récupération de mot de passe, etc.).
 - **Validation des données** : S'assurer que toutes les données entrantes (inscription, connexion, réinitialisation de mot de passe) sont validées pour éviter les erreurs et les failles de sécurité.

Résumé des tâches pour la gestion des utilisateurs et authentification

Tâche	Description
Inscription des utilisateurs	Création de comptes pour élèves, enseignants, parents et administrateurs.
Authentification avec JWT	Connexion sécurisée avec génération et validation de tokens JWT.
Gestion des rôles et permissions	Attribution des rôles et vérification des permissions pour chaque utilisateur.
Récupération de mot de passe	Système de réinitialisation de mot de passe via email.
Sécurité des données	Chiffrement des mots de passe et protection contre les attaques.
API pour la gestion des utilisateurs	Création des endpoints API pour l'inscription, la connexion, et la gestion des profils.
Tests et validation	Tests unitaires et d'intégration pour valider les fonctionnalités.

```
from rest_framework import serializers
from .models import CustomUser
from django.contrib.auth import authenticate
from rest_framework_simplejwt.tokens import RefreshToken
from rest_framework.response import Response
from rest_framework.views import APIView
# from .serializers import UserLoginSerializer
class UserLoginView(APIView):
    def post(self, request):
        serializer = UserLoginSerializer(data=request.data)
        if serializer.is_valid():
            user = serializer.validated_data
            refresh = RefreshToken.for_user(user)
            return Response({ 'refresh': str(refresh), 'access': str(refresh.access_token), })
        return Response(serializer.errors, status=400)
class UserRegistrationSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True)
    class Meta:
        model = CustomUser
        fields = ['username', 'email', 'password', 'role']
    def create(self, validated_data):
        user = User.objects.create_user(
            username=validated_data['username'],
            email=validated_data['email'],
            password=validated_data['password'],
            role=validated_data['role']
        )
        return user
class UserLoginSerializer(serializers.Serializer):
    email = serializers.EmailField()
    password = serializers.CharField(write_only=True)
    def validate(self, data):
        user = authenticate(email=data['email'], password=data['password'])
        if user and user.is_active:
            return user
        raise serializers.ValidationError("Identifiants incorrects")
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomUser
        fields = ['id', 'username', 'email', 'role']
class RegisterSerializer(serializers.ModelSerializer):
    password =
```

```
serializers.CharField(write_only=True) class Meta: model = CustomUser fields = ['id', 'username',
'email', 'password', 'role'] def create(self, validated_data): user = CustomUser.objects.create_user(
username=validated_data['username'], email=validated_data['email'],
password=validated_data['password'], role=validated_data['role'] ) return user from
rest_framework import serializers from django.contrib.auth import get_user_model from
rest_framework_simplejwt.serializers import TokenObtainPairSerializer User = get_user_model()
class CustomUserSerializer(serializers.ModelSerializer): class Meta: model = User fields = ('id',
'username', 'email', 'role', 'password') extra_kwargs = {'password': {'write_only': True}} def
create(self, validated_data): user = User.objects.create_user(**validated_data) return user class
CustomTokenObtainPairSerializer(TokenObtainPairSerializer): @classmethod def get_token(cls,
user): token = super().get_token(user) token['role'] = user.role return token
```

Le code que vous avez fourni est bien structuré et couvre les principales fonctionnalités pour la gestion des utilisateurs, l'authentification et la génération de tokens JWT. Cependant, il y a quelques points à améliorer ou à clarifier pour que le code soit plus cohérent et fonctionnel. Voici une analyse détaillée et des suggestions d'amélioration :

```
from rest_framework import serializers from .models import CustomUser from django.contrib.auth
import authenticate from rest_framework_simplejwt.tokens import RefreshToken from
rest_framework.response import Response from rest_framework.views import APIView # from
.serializers import UserLoginSerializer class UserLoginView(APIView): def post(self, request):
serializer = UserLoginSerializer(data=request.data) if serializer.is_valid(): user =
serializer.validated_data refresh = RefreshToken.for_user(user) return Response({'refresh':
str(refresh), 'access': str(refresh.access_token), }) return Response(serializer.errors, status=400)
class UserRegistrationSerializer(serializers.ModelSerializer): password =
serializers.CharField(write_only=True) class Meta: model = CustomUser fields = ['username',
'email', 'password', 'role'] def create(self, validated_data): user = User.objects.create_user(
username=validated_data['username'], email=validated_data['email'],
password=validated_data['password'], role=validated_data['role'] ) return user class
UserLoginSerializer(serializers.Serializer): email = serializers.EmailField() password =
serializers.CharField(write_only=True) def validate(self, data): user =
authenticate(email=data['email'], password=data['password']) if user and user.is_active: return
user raise serializers.ValidationError("Identifiants incorrects") class
UserSerializer(serializers.ModelSerializer): class Meta: model = CustomUser fields = ['id',
'username', 'email', 'role'] class RegisterSerializer(serializers.ModelSerializer): password =
serializers.CharField(write_only=True) class Meta: model = CustomUser fields = ['id', 'username',
'email', 'password', 'role'] def create(self, validated_data): user = CustomUser.objects.create_user(
username=validated_data['username'], email=validated_data['email'],
password=validated_data['password'], role=validated_data['role'] ) return user from
rest_framework import serializers from django.contrib.auth import get_user_model from
rest_framework_simplejwt.serializers import TokenObtainPairSerializer User = get_user_model()
class CustomUserSerializer(serializers.ModelSerializer): class Meta: model = User fields = ('id',
'username', 'email', 'role', 'password') extra_kwargs = {'password': {'write_only': True}} def
create(self, validated_data): user = User.objects.create_user(**validated_data) return user class
CustomTokenObtainPairSerializer(TokenObtainPairSerializer): @classmethod def get_token(cls,
user): token = super().get_token(user) token['role'] = user.role return token
```

Le code que vous avez fourni est bien structuré, mais il y a quelques problèmes et redondances qui peuvent être améliorés. Voici une version optimisée et corrigée de votre code, avec des explications pour chaque partie.

```
from django.db import models
from django.contrib.auth.models import AbstractUser
from django.utils import timezone
from django.core.exceptions import ValidationError
from datetime import timedelta
from django.storage import SecureFileStorage
from django.contrib.auth.models import License

class User(AbstractUser):
    ROLE_CHOICES = [
        ('student', 'Étudiant'),
        ('teacher', 'Professeur'),
        ('parent', 'Parent'),
        ('admin', 'Administrateur'),
    ]
    # first_name = models.CharField(max_length=100)
    # last_name = models.CharField(max_length=100)
    role = models.CharField(max_length=10, choices=ROLE_CHOICES, default='student')
    created_at = models.DateTimeField(default=timezone.now)
    fcm_token = models.CharField(max_length=255, blank=True, null=True)

    def __str__(self):
        return self.username

    def authenticate(self):
        """Authentifie l'utilisateur"""
        return self.is_authenticated

    def update_profile(self, **kwargs):
        """Met à jour le profil de l'utilisateur"""
        for key, value in kwargs.items():
            setattr(self, key, value)
        self.save()

class Student(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    grade = models.CharField(max_length=50)
    level = models.IntegerField(default=1)
    points = models.IntegerField(default=0)
    total_exercises_completed = models.IntegerField(default=0)
    streak_days = models.IntegerField(default=0)
    last_activity_date = models.DateField(null=True, blank=True)

    def add_points(self, points):
        """Ajoute des points à l'étudiant"""
        self.points += points
        self.save()

    from .rewards import calculate_level, check_and_award_badges
    # Met à jour le niveau si nécessaire
    new_level = calculate_level(self.points)
    if new_level > self.level:
        self.level = new_level
        self.save()
    # Vérifie les badges
    check_and_award_badges(self)

    def update_streak(self):
        """Met à jour la série de jours d'activité"""
        today = timezone.now().date()
        if self.last_activity_date == today:
            return
        if self.last_activity_date == today - timezone.timedelta(days=1):
            self.streak_days += 1
        else:
            self.streak_days = 1
        self.last_activity_date = today
        self.save()

    def get_progress_to_next_level(self):
        """Obtient la progression vers le niveau suivant"""
        from .rewards import calculate_progress_to_next_level
        return calculate_progress_to_next_level(self.points)

    def get_recent_achievements(self):
        """Obtient les réalisations récentes"""
        return self.badges.order_by('-id')[:5]

    def get_statistics(self):
        """Obtient les statistiques de l'étudiant"""
        return {
            'total_points': self.points,
            'current_level': self.level,
            'progress_to_next_level': self.get_progress_to_next_level(),
            'exercises_completed': self.total_exercises_completed,
            'streak_days': self.streak_days,
            'badges_count': self.badges.count(),
            'recent_achievements': [
                {
                    'name': badge.name,
                    'description': badge.description,
                    'image_url': badge.image.url if badge.image else None
                }
                for badge in self.get_recent_achievements()
            ]
        }

    def submit_exercise(self, exercise, answers):
        """Soumet un exercice pour évaluation"""
        # Logique pour soumettre et évaluer un exercice
        score = exercise.evaluate(answers)
        Progress.objects.create(
            student=self,
            course=exercise.course,
            score=score
        )
        return score

    def access_course(self, course):
        """Accède à un cours"""
        if course in self.classes.all().values_list('courses', flat=True):
            return True
        raise ValidationError("Vous n'avez pas accès à ce cours")

    def download_content(self, resource):
        """Télécharge une ressource"""
        if resource.offline_available:
            return resource.url
        raise ValidationError("Cette ressource n'est pas disponible hors ligne")

class Teacher(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)

    def create_exam(self, **kwargs):
        """Crée un nouvel examen"""
        return Exam.objects.create(**kwargs)

    def evaluate_student(self, student, course):
        """Évalue un étudiant"""
        progress = Progress.objects.filter(
            student=student,
            course=course
        )
        return progress.aggregate(models.Avg('score'))['score_avg']

    def create_content(self, course, **kwargs):
        """Crée du contenu pour un cours"""
        return Resource.objects.create(course=course, **kwargs)

    def manage_course(self, course, **kwargs):
        """Gère un cours"""
        for key, value in kwargs.items():
```

```

setattr(course, key, value) course.save() class Parent(models.Model): user =
models.OneToOneField(User, on_delete=models.CASCADE) children =
models.ManyToManyField(Student, related_name='parents') def view_progress(self, child):
"""Consulte la progression d'un enfant""" if child in self.children.all(): return
Progress.objects.filter(student=child) raise ValidationError("Vous n'avez pas accès aux
informations de cet étudiant") def receive_notifications(self): """Reçoit les notifications""" return
Notification.objects.filter(recipient=self.user) def communicate_with_teacher(self, teacher,
message): """Communique avec un professeur""" Notification.objects.create( type='info',
message=message, recipient=teacher.user ) class Admin(models.Model): user =
models.OneToOneField(User, on_delete=models.CASCADE) def manage_users(self): """Gère les
utilisateurs""" return User.objects.all() def moderate_content(self, content): """Modère le
contenu""" content.is_approved = True content.save() def generate_reports(self): """Génère des
rapports""" # Logique pour générer des rapports pass def configure_system(self, **settings):
"""Configure le système""" for key, value in settings.items(): # Logique pour mettre à jour les
paramètres système pass class Course(models.Model): title = models.CharField(max_length=200)
subject = models.CharField(max_length=100) description = models.TextField() level =
models.CharField(max_length=50) created_at = models.DateTimeField(auto_now_add=True)
teacher = models.ForeignKey(Teacher, on_delete=models.CASCADE, related_name='courses') #
Ajoutez les champs pour les fichiers multimédias video_file =
models.FileField(upload_to='videos/', blank=True, null=True) pdf_file =
models.FileField(upload_to='pdfs/', blank=True, null=True) audio_file =
models.FileField(upload_to='audios/', blank=True, null=True) def __str__(self): return self.title def
create_lesson(self, **kwargs): """Crée une nouvelle leçon""" return
Resource.objects.create(course=self, **kwargs) def update_content(self, **kwargs): """Met à jour le
contenu du cours""" for key, value in kwargs.items(): setattr(self, key, value) self.save() class
Resource(models.Model): TYPE_CHOICES = [ ('pdf', 'PDF'), ('video', 'Vidéo'), ('audio', 'Audio'),
('other', 'Autre'), ] course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='resources') type = models.CharField(max_length=10, choices=TYPE_CHOICES) url
= models.URLField() title = models.CharField(max_length=200) offline_available =
models.BooleanField(default=False) # ... vos champs existants ... file = models.FileField(
upload_to='resources/%Y/%m/%d/', storage=SecureFileStorage(), null=True, blank=True )
file_size = models.IntegerField(default=0) download_count = models.IntegerField(default=0)
last_downloaded = models.DateTimeField(null=True, blank=True) def save(self, *args, **kwargs): if
self.file: self.file_size = self.file.size super().save(*args, **kwargs) def increment_download(self):
self.download_count += 1 self.last_downloaded = timezone.now() self.save() def download(self):
"""Télécharge la ressource""" return self.url def stream(self): """Diffuse la ressource""" if self.type in
['video', 'audio']: return self.url raise ValidationError("Cette ressource ne peut pas être diffusée")
class Exercise(models.Model): course = models.ForeignKey(Course, on_delete=models.CASCADE,
related_name='exercises') title = models.CharField(max_length=200) type =
models.CharField(max_length=50) difficulty = models.IntegerField() points = models.IntegerField()
created_at = models.DateTimeField(auto_now_add=True) def evaluate(self, answers): """Évalue les
réponses de l'exercice""" # Logique d'évaluation return 0 # À implémenter selon les critères
d'évaluation def submit_answer(self, student, answers): """Soumet une réponse""" score =
self.evaluate(answers) return Progress.objects.create( student=student, course=self.course,
score=score ) class Progress(models.Model): student = models.ForeignKey(Student,
on_delete=models.CASCADE, related_name='progress') course = models.ForeignKey(Course,
on_delete=models.CASCADE) score = models.FloatField() completed_at =
models.DateTimeField(auto_now=True) def update_progress(self, new_score): """Met à jour la
progression""" self.score = new_score self.save() def generate_report(self): """Génère un rapport
de progression""" return { 'student': self.student.user.get_full_name(), 'course': self.course.title,

```



```

'score': self.score, 'completed_at': self.completed_at } class Class(models.Model): name =
models.CharField(max_length=100) teacher = models.ForeignKey(Teacher,
on_delete=models.CASCADE, related_name='classes') students =
models.ManyToManyField(Student, related_name='classes') courses =
models.ManyToManyField(Course, related_name='classes') def add_student(self, student):
"""Ajoute un étudiant à la classe""" self.students.add(student) def remove_student(self, student):
"""Retire un étudiant de la classe""" self.students.remove(student) def assign_course(self, course):
"""Assigne un cours à la classe""" self.courses.add(course) class Badge(models.Model): name =
models.CharField(max_length=100) description = models.TextField() image =
models.ImageField(upload_to='badges/') required_points = models.IntegerField() students =
models.ManyToManyField(Student, related_name='badges') def award(self, student): """Décérne
le badge à un étudiant""" if student.points >= self.required_points: self.students.add(student)
return True return False def check_eligibility(self, student): """Vérifie l'éligibilité d'un étudiant pour
le badge""" return student.points >= self.required_points class Notification(models.Model):
TYPE_CHOICES = [ ('info', 'Information'), ('success', 'Succès'), ('warning', 'Avertissement'), ('error',
'Erreur'), ] type = models.CharField(max_length=10, choices=TYPE_CHOICES) message =
models.TextField() created_at = models.DateTimeField(auto_now_add=True) recipient =
models.ForeignKey(User, on_delete=models.CASCADE, related_name='notifications') read =
models.BooleanField(default=False) def send(self): """Envoie la notification""" # Logique d'envoi
(email, push, etc.) pass def mark_as_read(self): """Marque la notification comme lue""" self.read =
True self.save() class Exam(models.Model): from .exam_management import ExamSession
EXAM_TYPES = [ ('practice', 'Examen Blanc'), ('final', 'Examen Final'), ('quiz', 'Quiz'), ] title =
models.CharField(max_length=200) type = models.CharField(max_length=20,
choices=EXAM_TYPES, default='practice') date = models.DateTimeField() duration =
models.DurationField(default=timedelta(hours=2)) total_points = models.FloatField() course =
models.ForeignKey(Course, on_delete=models.CASCADE, related_name='exams') exercises =
models.ManyToManyField(Exercise, related_name='exams') is_practice =
models.BooleanField(default=False) passing_score = models.FloatField(default=60.0) instructions =
models.TextField(blank=True) allowed_attempts = models.IntegerField(default=1) class Meta:
ordering = ['-date'] def get_active_sessions(self): """Récupère les sessions d'examen actives"""
from .exam_management import ExamSession return ExamSession.objects.filter( exam=self,
end_time__isnull=True ) def start_session(self, student): """Démontre une nouvelle session
d'examen""" from .exam_management import ExamSession return ExamSession(self, student) def
grade_exam(self, student_answers): """Note l'examen complet""" total_score = 0 results = [] for
exercise, answer in student_answers.items(): score = exercise.evaluate(answer) total_score +=
score * exercise.points results.append({ 'exercise': exercise, 'score': score, 'max_points':
exercise.points }) final_score = (total_score / self.total_points) * 100 return { 'score': final_score,
'passed': final_score >= self.passing_score, 'details': results } def get_statistics(self): """Obtient les
statistiques de l'examen""" from .exam_management import analyze_exam_results return
analyze_exam_results(self) def schedule_for_class(self, class_group): """Planifie l'examen pour une
classe""" from .exam_management import schedule_exam schedule_exam(self, class_group) def
generate_student_report(self, student): """Génère un rapport détaillé pour un étudiant""" from
.exam_management import generate_exam_report return generate_exam_report(self, student) #
class License(models.Model): # LICENSE_TYPES = [ # ('basic', 'Basique'), # ('premium', 'Premium'),
# ('school', 'École'), # ] # STATUS_CHOICES = [ # ('active', 'Active'), # ('expired', 'Expirée'), #
('cancelled', 'Annulée'), # ('pending', 'En attente'), # ] # type = models.CharField(max_length=20,
choices=LICENSE_TYPES) # user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='licenses') # start_date = models.DateTimeField(auto_now_add=True) # end_date =
models.DateTimeField() # status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='pending') # max_students = models.IntegerField(default=1) # Pour les licences école #

```

```

features = models.JSONField(default=dict) # Stocke les fonctionnalités activées # class Meta: #
ordering = ['-start_date'] # def is_valid(self): # """Vérifie si la licence est valide""" # now =
timezone.now() # return ( # self.status == 'active' and # self.start_date <= now <= self.end_date #
) # def get_remaining_days(self): # """Retourne le nombre de jours restants""" # if not
self.is_valid(): # return 0 # now = timezone.now() # return (self.end_date - now).days # def
renew(self, duration_months=1): # """Renouvelle la licence""" # if self.status not in ['active',
'expired']: # raise ValidationError("Impossible de renouveler cette licence") # self.start_date =
timezone.now() # self.end_date = self.start_date + timedelta(days=30 * duration_months) #
self.status = 'active' # self.save() class Payment(models.Model): from keysGen.models import
License PAYMENT_TYPES = [ ('mobile', 'Païement Mobile'), ('card', 'Carte Bancaire'), ('transfer',
'Virement Bancaire'), ] STATUS_CHOICES = [ ('pending', 'En attente'), ('completed', 'Complété'),
('failed', 'Échoué'), ('refunded', 'Remboursé'), ] user = models.ForeignKey(User,
on_delete=models.CASCADE, related_name='payments') license = models.ForeignKey(License,
on_delete=models.CASCADE, related_name='payments') amount =
models.DecimalField(max_digits=10, decimal_places=2) currency =
models.CharField(max_length=3, default='XAF') payment_type =
models.CharField(max_length=20, choices=PAYMENT_TYPES) status =
models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending') transaction_id =
models.CharField(max_length=100, unique=True) payment_date =
models.DateTimeField(auto_now_add=True) class Meta: ordering = ['-payment_date'] def
process_payment(self): """Traite le paiement""" from .payment_processing import
process_payment success = process_payment(self) if success: self.status = 'completed'
self.license.status = 'active' self.license.save() else: self.status = 'failed' self.save() return success def
generate_receipt(self): """Génère un reçu de paiement""" from .payment_processing import
generate_receipt return generate_receipt(self) class Subscription(models.Model): from
keysGen.models import License """Gère les abonnements récurrents""" INTERVAL_CHOICES = [
('monthly', 'Mensuel'), ('quarterly', 'Trimestriel'), ('yearly', 'Annuel'), ] user =
models.ForeignKey(User, on_delete=models.CASCADE, related_name='subscriptions') license =
models.ForeignKey(License, on_delete=models.CASCADE, related_name='subscription') interval =
models.CharField(max_length=20, choices=INTERVAL_CHOICES) auto_renew =
models.BooleanField(default=True) next_billing_date = models.DateTimeField() def
process_renewal(self): """Traite le renouvellement automatique""" if not self.auto_renew: return
False payment = Payment.objects.create( user=self.user, license=self.license,
amount=self.get_renewal_amount(), payment_type=self.user.preferred_payment_method ) success
= payment.process_payment() if success: self.update_next_billing_date() return success def
update_next_billing_date(self): """Met à jour la prochaine date de facturation""" intervals = {
'monthly': 1, 'quarterly': 3, 'yearly': 12 } months = intervals.get(self.interval, 1) self.next_billing_date
= timezone.now() + timedelta(days=30 * months) self.save() def get_renewal_amount(self):
"""Calcule le montant du renouvellement""" from .payment_processing import
calculate_subscription_amount return calculate_subscription_amount(self.license.type, self.interval)
class MobilePayment(models.Model): PROVIDER_CHOICES = [ ('orange', 'Orange Money'), ('mtn',
'MTN Mobile Money'), ] STATUS_CHOICES = [ ('initiated', 'Initié'), ('pending', 'En attente'),
('completed', 'Complété'), ('failed', 'Échoué'), ('cancelled', 'Annulé'), ] payment =
models.OneToOneField( Payment, on_delete=models.CASCADE, related_name='mobile_payment'
) provider = models.CharField(max_length=20, choices=PROVIDER_CHOICES) phone_number =
models.CharField(max_length=20) reference = models.CharField(max_length=100, unique=True)
status = models.CharField( max_length=20, choices=STATUS_CHOICES, default='initiated' )
callback_url = models.URLField() callback_data = models.JSONField(default=dict) initiated_at =
models.DateTimeField(auto_now_add=True) completed_at = models.DateTimeField(null=True,
blank=True) class Meta: ordering = ['-initiated_at'] def initiate_payment(self): """Initie le paiement

```

```
mobile""" from .payment_providers import initiate_mobile_payment success, reference =
initiate_mobile_payment(self) if success: self.reference = reference self.status = 'pending'
self.save() return success def check_status(self): """Vérifie le statut du paiement""" from
.payment_providers import check_payment_status status = check_payment_status(self) if status !=
self.status: self.status = status if status == 'completed': self.completed_at = timezone.now()
self.payment.status = 'completed' self.payment.save() elif status in ['failed', 'cancelled']:
self.payment.status = 'failed' self.payment.save() self.save() return status def process_callback(self,
data): """Traite le callback du fournisseur de paiement""" self.callback_data = data status =
data.get('status') if status in dict(self.STATUS_CHOICES): self.status = status if status ==
'completed': self.completed_at = timezone.now() self.payment.status = 'completed'
self.payment.save() elif status in ['failed', 'cancelled']: self.payment.status = 'failed'
self.payment.save() self.save() return self.status class MobilePaymentConfig(models.Model):
"""Configuration des fournisseurs de paiement mobile""" PROVIDER_CHOICES = [ ('orange',
'Orange Money'), ('mtn', 'MTN Mobile Money'), ] provider = models.CharField( max_length=20,
choices=PROVIDER_CHOICES, unique=True ) merchant_id = models.CharField(max_length=100)
api_key = models.CharField(max_length=100) api_secret = models.CharField(max_length=100)
api_url = models.URLField() callback_url = models.URLField() is_active =
models.BooleanField(default=True) test_mode = models.BooleanField(default=False) def
get_credentials(self): """Retourne les informations d'authentification""" return { 'merchant_id':
self.merchant_id, 'api_key': self.api_key, 'api_secret': self.api_secret, 'api_url': self.api_url,
'callback_url': self.callback_url, 'test_mode': self.test_mode }
```

Votre projet est très complet et bien structuré ! Il couvre une grande variété de fonctionnalités, allant de la gestion des utilisateurs à la gestion des paiements, en passant par les cours, les exercices, les badges, et les notifications. Voici une analyse détaillée de votre projet, ainsi que des suggestions pour l'améliorer et le compléter.

```
from django.urls import path, include from django.contrib.auth.views import LogoutView from .
import views urlpatterns = [ path('api/', include('core.api.urls')), path('', views.home,
name='home'), path('payment/', views.payment, name='payment'), path('paymentMobile/',
views.paymentMobile, name='paymentMobile'), path('login/', views.login_view, name='login'),
path('logout/', LogoutView.as_view(next_page='home'), name='logout'), path('register/',
views.register, name='register'), path('receipts_home/', views.receipts_home,
name='receipts_home'), path('courses/', views.course_list, name='course_list'), path('courses/add/',
views.course_create, name='course_create'), path('courses/edit/<int:course_id>/',
views.course_edit, name='course_edit'), path('courses/<int:course_id>/', views.course_detail,
name='course_detail'), path('register-fcm-token/', views.register_fcm_token,
name='register_fcm_token'), ]
```

Votre fichier `urls.py` est bien structuré et couvre les principales routes de votre application. Voici une analyse détaillée de vos routes et des suggestions pour les améliorer ou les compléter.

```
import token from django import forms from django.http import response from django.shortcuts
import render, redirect, get_object_or_404 from django.utils.ipv6 import is_valid_ipv6_address
from .models import Payment, Teacher, User from django.contrib.auth.decorators import
login_required from django.contrib.auth import authenticate, login from
django.contrib.auth.forms import AuthenticationForm from django.contrib.auth.forms import
UserCreationForm from . import forms from django.contrib import messages from firebase_admin
```

```

import messaging from django.http import JsonResponse @login_required def
register_fcm_token(request): if request.method == 'POST': token = request.POST.get('token') if
token: request.user.fcm_token = token request.user.save() return JsonResponse({'status':
'success'}) return JsonResponse({'status': 'error', 'message': 'Token maquant'}) return
JsonResponse({'status': 'error', 'message': 'Méthode non autorisée'}) def
send_push_notification(token, title, body): message = messaging.Message(
notification=messaging.Notification( title=title, body=body ), token=token ) response =
messaging.send(message) return response def register(request): if request.method == 'POST':
form = forms.SignUpForm(request.POST) if form.is_valid(): if
User.objects.filter(email=form.cleaned_data['email']).exists(): messages.error(request, 'Un compte
existe déjà avec cette adresse email') return redirect('register') user = form.save(commit=False) #
Ne pas enregistrer tout de suite user.first_name = form.cleaned_data['first_name'] user.last_name
= form.cleaned_data['last_name'] user.username = form.cleaned_data['username'] user.email =
form.cleaned_data['email'] user.save() login(request, user) return redirect('login') else: form =
forms.SignUpForm() return render(request, 'register.html', {'form': form}) def login_view(request): if
request.method == 'POST': form = AuthenticationForm(request, data=request.POST) if
form.is_valid(): username = form.cleaned_data.get('username') password =
form.cleaned_data.get('password') user = authenticate(username=username,
password=password) if user is not None: login(request, user) return redirect('receipts_home') #
Rediriger vers la page d'accueil après la connexion else: form = AuthenticationForm() return
render(request, 'login.html', {'form': form}) def home(request): return render(request, 'home.html')
def paymentMobile(request): return render(request, 'paymentMobile.html') @login_required def
payment(request): if request.method == 'POST': amount = request.POST.get('amount') # Créer un
nouvel enregistrement de paiement Payment.objects.create(user=request.user, amount=amount,
status='pending') return redirect('home') # Rediriger vers la page d'accueil après la création
return render(request, 'payment.html') def receipts_home(request): return render(request,
'receipts_home.html') from .models import Course def course_list(request): courses =
Course.objects.all() return render(request, 'course_list.html', {'courses': courses}) # core/views.py
@login_required def course_create(request): if request.method == 'POST': form =
forms.CourseForm(request.POST, request.FILES) if form.is_valid(): course =
form.save(commit=False) try: course.teacher = Teacher.objects.get(user=request.user) #
Récupérez l'enseignant except Teacher.DoesNotExist: # Gérer le cas où l'utilisateur n'est pas un
enseignant return render(request, 'course_create.html', {'form': form, 'error': 'Vous devez être un
enseignant pour créer un cours.}) course.save() return redirect('course_list') else: form =
forms.CourseForm() return render(request, 'course_create.html', {'form': form}) def
course_edit(request, course_id): course = Course.objects.get(id=course_id) if request.method ==
'POST': form = forms.CourseForm(request.POST, instance=course) if form.is_valid(): form.save()
return redirect('course_list') else: form = forms.CourseForm(instance=course) return
render(request, 'course_create.html', {'form': form}) @login_required def course_detail(request,
course_id): course = get_object_or_404(Course, id=course_id) return render(request,
'course_detail.html', {'course': course})

```

Votre fichier views.py est bien structuré et couvre une grande variété de fonctionnalités, allant de l'authentification à la gestion des cours, en passant par les paiements et les notifications push. Voici une analyse détaillée de votre code, ainsi que des suggestions pour l'améliorer et le compléter.

```

from django.core.files.storage import FileSystemStorage from django.conf import settings import
os import hashlib class SecureFileStorage(FileSystemStorage): def __init__(self):

```

```

super().__init__(location=settings.MEDIA_ROOT)
def get_valid_name(self, name): """Génère un nom de fichier sécurisé"""
    ext = os.path.splitext(name)[1]
    name_hash = hashlib.md5(name.encode()).hexdigest()
    return f"{name_hash}{ext}"
def get_available_name(self, name, max_length=None):
    """S'assure que le nom du fichier est unique"""
    if self.exists(name):
        dir_name, file_name = os.path.split(name)
        file_root, file_ext = os.path.splitext(file_name)
        counter = 1
        while self.exists(name):
            name = os.path.join(dir_name, f"{file_root}_{counter}{file_ext}")
            counter += 1
        return name

```

Votre classe `SecureFileStorage` est une excellente implémentation d'un système de stockage de fichiers sécurisé. Elle permet de générer des noms de fichiers uniques et sécurisés en utilisant un hachage MD5, ce qui est une bonne pratique pour éviter les collisions de noms de fichiers et renforcer la sécurité. Voici une analyse détaillée de votre code et quelques suggestions pour l'améliorer.

```

from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Student, Progress, Badge, Exercise

# Définition des niveaux et points requis
LEVEL_THRESHOLDS = { 1: 0, 2: 100, 3: 250, 4: 500, 5: 1000, 6: 2000, 7: 3500, 8: 5000, 9: 7500, 10: 10000 }

# Définition des badges par défaut
DEFAULT_BADGES = [
    { 'name': 'Débutant', 'description': 'Premier pas dans l\'apprentissage', 'required_points': 0, 'image': 'badges/beginner.png' },
    { 'name': 'Apprenti', 'description': 'Progression constante', 'required_points': 500, 'image': 'badges/apprentice.png' },
    { 'name': 'Expert', 'description': 'Maîtrise avancée', 'required_points': 2000, 'image': 'badges/expert.png' },
    { 'name': 'Maître', 'description': 'Excellence académique', 'required_points': 5000, 'image': 'badges/master.png' }
]

def calculate_level(points):
    """Calcule le niveau en fonction des points"""
    for level, threshold in sorted(LEVEL_THRESHOLDS.items(), reverse=True):
        if points >= threshold:
            return level
    return 1

def calculate_progress_to_next_level(points):
    """Calcule la progression vers le niveau suivant"""
    current_level = calculate_level(points)
    next_level = current_level + 1
    if next_level > max(LEVEL_THRESHOLDS.keys()):
        return 100
    current_threshold = LEVEL_THRESHOLDS[current_level]
    next_threshold = LEVEL_THRESHOLDS[next_level]
    progress = ((points - current_threshold) / (next_threshold - current_threshold)) * 100
    return min(progress, 100)

@receiver(post_save, sender=Progress)
def update_student_points(sender, instance, created, **kwargs):
    """Met à jour les points de l'étudiant après une progression"""
    if created:
        student = instance.student
        # Ajoute les points basés sur le score
        points_earned = int(instance.score * instance.exercise.points / 100)
        student.points += points_earned
        # Met à jour le niveau
        new_level = calculate_level(student.points)
        if new_level > student.level:
            student.level = new_level
        student.save()
    # Vérifie les badges
    check_and_award_badges(student)

def check_and_award_badges(student):
    """Vérifie et attribue les badges si l'étudiant est éligible"""
    available_badges = Badge.objects.all()
    for badge in available_badges:
        if badge.check_eligibility(student) and not student.badges.filter(id=badge.id).exists():
            badge.award(student)

```

Votre code utilise des signaux Django (`post_save`) pour mettre à jour les points, les niveaux et les badges des étudiants lorsqu'une progression est enregistrée. C'est une excellente approche pour automatiser ces mises à jour. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```

import requests
import uuid
from datetime import datetime
from django.conf import settings
def get_provider_config(provider):
    """Récupère la configuration du fournisseur de paiement"""
    from

```



```

.models import MobilePaymentConfig try: return MobilePaymentConfig.objects.get(
provider=provider, is_active=True ) except MobilePaymentConfig.DoesNotExist: return None def
generate_reference(): """Génère une référence unique pour le paiement""" return f"PAY-
{datetime.now().strftime('%Y%m%d')}-{uuid.uuid4().hex[:8]}" def
initiate_mobile_payment(payment): """Initie un paiement mobile avec le fournisseur approprié"""
config = get_provider_config(payment.provider) if not config: return False, None if
payment.provider == 'orange': return initiate_orange_money_payment(payment, config) elif
payment.provider == 'mtn': return initiate_mtn_money_payment(payment, config) return False,
None def check_payment_status(payment): """Vérifie le statut d'un paiement""" config =
get_provider_config(payment.provider) if not config: return 'failed' if payment.provider ==
'orange': return check_orange_money_status(payment, config) elif payment.provider == 'mtn':
return check_mtn_money_status(payment, config) return 'failed' def
initiate_orange_money_payment(payment, config): """Initie un paiement Orange Money"""
reference = generate_reference() headers = { 'Authorization': f'Bearer {config.api_key}', 'Content-
Type': 'application/json' } payload = { 'merchant_id': config.merchant_id, 'reference': reference,
'amount': str(payment.payment.amount), 'currency': payment.payment.currency, 'phone_number':
payment.phone_number, 'callback_url': f'{config.callback_url}/orange/{reference}/', 'description':
f"Paiement {payment.payment.license.type}" } try: response = requests.post(
f"{config.api_url}/payment/initiate", json=payload, headers=headers ) if response.status_code ==
200: return True, reference return False, None except requests.exceptions.RequestException:
return False, None def initiate_mtn_money_payment(payment, config): """Initie un paiement MTN
Mobile Money""" reference = generate_reference() headers = { 'X-API-Key': config.api_key,
'Content-Type': 'application/json' } payload = { 'merchant': config.merchant_id, 'reference':
reference, 'amount': str(payment.payment.amount), 'currency': payment.payment.currency,
'phone': payment.phone_number, 'callback': f'{config.callback_url}/mtn/{reference}/', 'description':
f"Paiement {payment.payment.license.type}" } try: response = requests.post(
f"{config.api_url}/collections/initiate", json=payload, headers=headers ) if response.status_code
== 200: return True, reference return False, None except requests.exceptions.RequestException:
return False, None def check_orange_money_status(payment, config): """Vérifie le statut d'un
paiement Orange Money""" headers = { 'Authorization': f'Bearer {config.api_key}', 'Content-Type':
'application/json' } try: response = requests.get(
f"{config.api_url}/payment/status/{payment.reference}", headers=headers ) if
response.status_code == 200: data = response.json() return data.get('status', 'failed') return 'failed'
except requests.exceptions.RequestException: return 'failed' def
check_mtn_money_status(payment, config): """Vérifie le statut d'un paiement MTN Mobile
Money""" headers = { 'X-API-Key': config.api_key, 'Content-Type': 'application/json' } try: response
= requests.get( f"{config.api_url}/collections/status/{payment.reference}", headers=headers ) if
response.status_code == 200: data = response.json() return data.get('status', 'failed') return 'failed'
except requests.exceptions.RequestException: return 'failed'

```

Votre code pour la gestion des paiements mobiles est bien structuré et couvre les principales fonctionnalités nécessaires pour initier et vérifier les paiements via Orange Money et MTN Mobile Money. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```

from decimal import Decimal from datetime import datetime, timedelta from
django.template.loader import render_to_string from django.conf import settings def
process_payment(payment): """ Traite un paiement selon son type """ if payment.payment_type
== 'mobile': return True # Le traitement est géré par le module payment_providers elif
payment.payment_type == 'card': return process_card_payment(payment) elif

```

```

payment.payment_type == 'transfer': return process_transfer_payment(payment) return False def
process_card_payment(payment): """ Traite un paiement par carte À implémenter avec un
processeur de paiement par carte """ return False def process_transfer_payment(payment): """
Traite un paiement par virement Le statut sera mis à jour manuellement """ return True def
calculate_subscription_amount(license_type, interval): """ Calcule le montant d'un abonnement
selon le type de licence et l'intervalle """ base_prices = { 'basic': Decimal('5000'), # 5,000 XAF par
mois 'premium': Decimal('10000'), # 10,000 XAF par mois 'school': Decimal('50000') # 50,000 XAF
par mois } interval_multipliers = { 'monthly': 1, 'quarterly': 2.7, # 10% de réduction 'yearly': 10 #
17% de réduction } base_price = base_prices.get(license_type, Decimal('5000')) multiplier =
Decimal(str(interval_multipliers.get(interval, 1))) return base_price * multiplier def
generate_receipt(payment): """ Génère un reçu de paiement au format HTML """ context = {
'payment': payment, 'user': payment.user, 'license': payment.license, 'company_name': 'eCEP',
'company_address': 'Yaoundé, Cameroun', 'company_phone': '+237 6XX XXX XXX',
'company_email': 'contact@ecep.cm', 'generated_at': datetime.now(), } html_content =
render_to_string('payment.html', context) return { 'html': html_content, 'payment_id': payment.id,
'amount': str(payment.amount), 'date': payment.payment_date.strftime('%Y-%m-%d %H:%M:%S') }
def send_payment_notification(payment): """ Envoie une notification de paiement à l'utilisateur """
from .models import Notification if payment.status == 'completed': message = f"Votre paiement
de {payment.amount} {payment.currency} a été confirmé." elif payment.status == 'failed': message
= f"Votre paiement de {payment.amount} {payment.currency} a échoué." else: return
Notification.objects.create( user=payment.user, type='payment', message=message, data={
'payment_id': payment.id, 'amount': str(payment.amount), 'status': payment.status } ) def
check_expired_subscriptions(): """ Vérifie et traite les abonnements expirés À exécuter
quotidiennement via une tâche cron """ from .models import Subscription today = datetime.now()
expiring_soon = today + timedelta(days=7) # Trouve les abonnements qui expirent bientôt
subscriptions = Subscription.objects.filter( next_billing_date__lte=expiring_soon, auto_renew=True
) for subscription in subscriptions: if subscription.next_billing_date <= today: # Tente le
renouvellement subscription.process_renewal() else: # Envoie un rappel
send_renewal_reminder(subscription) def send_renewal_reminder(subscription): """ Envoie un
rappel de renouvellement """ from .models import Notification days_left =
(subscription.next_billing_date - datetime.now()).days amount = calculate_subscription_amount(
subscription.license.type, subscription.interval ) message = ( f"Votre abonnement sera renouvelé
automatiquement dans {days_left} jours. " f"Montant : {amount} XAF" ) Notification.objects.create(
user=subscription.user, type='renewal_reminder', message=message, data={ 'subscription_id':
subscription.id, 'days_left': days_left, 'amount': str(amount) } ) import qrcode from io import
BytesIO from django.core.files.base import ContentFile def generate_qr_code(payment): """
Génère un QR code pour le paiement """ # Lien ou texte à encoder qr_data =
f"https://ecep.cm/verify/{payment.transaction_id}" # Créer le QR code qr = qrcode.QRCode(
version=1, error_correction=qrcode.constants.ERROR_CORRECT_L, box_size=10, border=4, )
qr.add_data(qr_data) qr.make(fit=True) # Créer une image du QR code img =
qr.make_image(fill_color="black", back_color="white") # Enregistrer l'image dans un buffer buffer
= BytesIO() img.save(buffer, format='PNG') buffer.seek(0) # Créer un fichier ContentFile pour
l'enregistrement qr_code_file = ContentFile(buffer.read(),
name=f"qr_code_{payment.transaction_id}.png") # Retourner le fichier QR code return
qr_code_file

```

Votre code pour la gestion des paiements, des abonnements, et des notifications est bien structuré et couvre une grande variété de fonctionnalités. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```

from django import forms
from django.contrib.auth.forms import UserCreationForm
from core.models import User
from keysGen.models import License
from django.core.exceptions import ValidationError

class LoginForm(forms.Form):
    username = forms.CharField(
        widget=forms.TextInput(
            attrs={
                'class': 'form-control',
                'placeholder': 'Entrez votre Nom'
            }
        ),
        password = forms.CharField(
            widget=forms.PasswordInput(
                attrs={
                    'class': 'form-control',
                    'placeholder': 'Entrez votre mot de passe'
                }
            )
        ),
        class SignUpForm(UserCreationForm):
            first_name = forms.CharField(
                max_length=30,
                required=True,
                widget=forms.TextInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Entrez votre prénom'
                    }
                )
            )
            last_name = forms.CharField(
                max_length=30,
                required=True,
                widget=forms.TextInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Entrez votre nom'
                    }
                )
            )
            username = forms.CharField(
                widget=forms.TextInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Entrez votre nom'
                    }
                )
            )
            email = forms.EmailField(
                widget=forms.EmailInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Entrez votre email'
                    }
                )
            )
            role = forms.ChoiceField(
                choices=User.ROLE_CHOICES,
                widget=forms.Select(
                    attrs={
                        'class': 'form-control'
                    }
                )
            )
            password1 = forms.CharField(
                widget=forms.PasswordInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Entrez votre mot de passe'
                    }
                )
            )
            password2 = forms.CharField(
                widget=forms.PasswordInput(
                    attrs={
                        'class': 'form-control',
                        'placeholder': 'Confirmez votre mot de passe'
                    }
                )
            )
            class Meta:
                model = User
                fields = ('first_name', 'last_name', 'username', 'email', 'role', 'password1', 'password2')

            def clean_username(self):
                username = self.cleaned_data.get('username')
                if User.objects.filter(username=username).exists():
                    raise ValidationError("Ce nom d'utilisateur est déjà pris.")
                return username

            class LicenseForm(forms.ModelForm):
                serial_number = forms.CharField(
                    max_length=17,
                    min_length=17,
                    widget=forms.TextInput(
                        attrs={
                            'class': 'form-control',
                            'placeholder': 'Entrez le numéro de série (17 chiffres)'
                        }
                    )
                )
                description = forms.CharField(
                    widget=forms.Textarea(
                        attrs={
                            'class': 'form-control',
                            'placeholder': 'Entrez la description'
                        }
                    )
                )
                class Meta:
                    model = License
                    fields = ['serial_number', 'description']

                def clean_serial_number(self):
                    serial_number = self.cleaned_data.get('serial_number')
                    if not serial_number.isdigit():
                        raise forms.ValidationError("Le numéro de série doit contenir uniquement des chiffres.")
                    if LicenseKey.objects.filter(serial_number=serial_number).exists():
                        raise forms.ValidationError("Ce numéro de série existe déjà.")
                    return serial_number

            from .models import Course
            class CourseForm(forms.ModelForm):
                class Meta:
                    model = Course
                    fields = ['title', 'description', 'level', 'video_file', 'pdf_file', 'audio_file']

```

Votre fichier `forms.py` est bien structuré et couvre les principales fonctionnalités nécessaires pour gérer l'inscription, la connexion, les licences et les cours. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```

from django.utils import timezone
from datetime import timedelta
from django.db.models import Avg, Count
from .models import Exercise, Student, Notification
class ExamSession:
    def __init__(self, exam, student):
        self.exam = exam
        self.student = student
        self.start_time = timezone.now()
        self.answers = {}

    def submit_answer(self, exercise_id, answer):
        """Enregistre la réponse à un exercice"""
        self.answers[exercise_id] = {
            'answer': answer,
            'time': timezone.now()
        }

    def get_remaining_time(self):
        """Calcule le temps restant pour l'examen"""
        elapsed = timezone.now() - self.start_time
        return max(timedelta(0), self.exam.duration - elapsed)

    def is_time_up(self):
        """Vérifie si le temps est écoulé"""
        return self.get_remaining_time() == timedelta(0)

    def create_practice_exam(self, subject, difficulty, num_exercises=10):
        """Crée un examen blanc basé sur le sujet et la difficulté"""
        exercises = Exercise.objects.filter(
            course__subject=subject,
            difficulty=difficulty
        ).order_by('?')[:num_exercises]
        if not exercises:
            return None
        exam = Exam.objects.create(
            title=f"Examen blanc - {subject}",
            date=timezone.now(),

```



```

duration=timedelta(minutes=30), total_points=sum(ex.points for ex in exercises), is_practice=True
) exam.exercises.set(exercises) return exam def analyze_exam_results(exam): from .models import
Exam, Progress """Analyse les résultats d'un examen""" results =
Progress.objects.filter(exercise__exams=exam) return { 'average_score':
results.aggregate(Avg('score'))['score__avg'] or 0, 'completion_rate': results.count() /
(exam.exercises.count() * Student.objects.count()) * 100, 'difficulty_distribution': results.values(
'exercise__difficulty' ).annotate( count=Count('id'), avg_score=Avg('score') ), 'time_distribution':
results.values( 'completion_time' ).annotate( count=Count('id') ) } def schedule_exam(exam,
class_group): """Planifie un examen pour une classe""" # Notifie les étudiants for student in
class_group.students.all(): Notification.objects.create( recipient=student.user, type='exam',
message=f"Nouvel examen prévu : {exam.title} le {exam.date}", ) # Crée des rappels
reminder_dates = [ exam.date - timedelta(days=7), exam.date - timedelta(days=1), exam.date -
timedelta(hours=1) ] for date in reminder_dates: if date > timezone.now():
Notification.objects.create( recipient=student.user, type='reminder', message=f"Rappel : Examen
{exam.title} prévu le {exam.date}", scheduled_for=date ) def generate_exam_report(exam, student):
"""Génère un rapport détaillé pour un étudiant""" progress_items = Progress.objects.filter(
student=student, exercise__exams=exam ) total_points = sum(p.score * p.exercise.points / 100 for
p in progress_items) max_points = exam.total_points return { 'student': { 'name':
student.user.get_full_name(), 'grade': student.grade }, 'exam': { 'title': exam.title, 'date': exam.date,
'duration': exam.duration }, 'results': { 'total_score': (total_points / max_points) * 100 if max_points
else 0, 'exercises_completed': progress_items.count(), 'total_exercises': exam.exercises.count(),
'time_spent': sum((p.completion_time for p in progress_items), timedelta()), 'exercise_details': [ {
'title': p.exercise.title, 'score': p.score, 'time_spent': p.completion_time, 'difficulty':
p.exercise.difficulty } for p in progress_items ] }, 'recommendations':
generate_recommendations(student, progress_items) } def generate_recommendations(student,
progress_items): """Génère des recommandations basées sur les performances""" weak_areas = []
for item in progress_items: if item.score < 60: weak_areas.append({ 'subject':
item.exercise.course.subject, 'topic': item.exercise.title, 'recommended_exercises':
Exercise.objects.filter( course__subject=item.exercise.course.subject,
difficulty__lte=item.exercise.difficulty ).exclude( progress__student=student )[:3] }) return {
'weak_areas': weak_areas, 'suggested_practice': bool(weak_areas), 'recommended_difficulty': max(
min(item.exercise.difficulty for item in progress_items if item.score < 70), 1 ) }

```

Votre code pour la gestion des examens, des sessions d'examen, et des rapports est bien structuré et couvre une grande variété de fonctionnalités. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```

from rest_framework import viewsets, permissions, status from rest_framework.decorators import
action from rest_framework.response import Response from rest_framework.views import APIView
from django.shortcuts import get_object_or_404 from core.exam_management import
ExamSession from .serializers import License, SubscriptionSerializer, ParentSerializer,
UserSerializer, ExamSessionSerializer, ExamStatisticsSerializer, AdminSerializer from core.models
import Parent, MobilePaymentConfig from core.models import ( User, Student, Teacher, Parent,
Admin, Course, Resource, Exercise, Progress, Class, Badge, Notification, Exam, #ExamSession,
Payment, Subscription, MobilePayment, ) from .serializers import ( UserSerializer,
StudentSerializer, TeacherSerializer, ParentSerializer, CourseSerializer, ResourceSerializer,
ExerciseSerializer, ProgressSerializer, ClassSerializer, BadgeSerializer, NotificationSerializer,
ExamSerializer, UserRegistrationSerializer, LicenseSerializer, PaymentSerializer,
SubscriptionSerializer, MobilePaymentSerializer, MobilePaymentConfigSerializer ) from

```

```

keysGen.models import License class MobilePaymentViewSet(viewsets.ModelViewSet):
serializer_class = MobilePaymentSerializer permission_classes = [permissions.IsAuthenticated] def
get_queryset(self): user = self.request.user if hasattr(user, 'admin'): return
MobilePayment.objects.all() return MobilePayment.objects.filter(payment__user=user) def
create(self, request, *args, **kwargs): """Crée et initie un nouveau paiement mobile""" serializer =
self.get_serializer(data=request.data) serializer.is_valid(raise_exception=True) # Crée le paiement
mobile mobile_payment = serializer.save() # Initie le paiement success =
mobile_payment.initiate_payment() if not success: mobile_payment.delete() return Response(
{'error': 'Échec de l\'initiation du paiement'}, status=status.HTTP_400_BAD_REQUEST ) headers =
self.get_success_headers(serializer.data) return Response( serializer.data,
status=status.HTTP_201_CREATED, headers=headers ) @action(detail=True, methods=['get']) def
check_status(self, request, pk=None): """Vérifie le statut d'un paiement""" mobile_payment =
self.get_object() current_status = mobile_payment.check_status() return Response({ 'status':
current_status, 'payment': MobilePaymentSerializer(mobile_payment).data }) @action(detail=True,
methods=['post']) def cancel(self, request, pk=None): """Annule un paiement en attente"""
mobile_payment = self.get_object() if mobile_payment.status not in ['initiated', 'pending']: return
Response( {'error': 'Ce paiement ne peut plus être annulé'},
status=status.HTTP_400_BAD_REQUEST ) mobile_payment.status = 'cancelled'
mobile_payment.save() mobile_payment.payment.status = 'failed' mobile_payment.payment.save()
return Response({ 'message': 'Paiement annulé', 'payment':
MobilePaymentSerializer(mobile_payment).data }) class
MobilePaymentConfigViewSet(viewsets.ModelViewSet): queryset =
MobilePaymentConfig.objects.all() serializer_class = MobilePaymentConfigSerializer class
ProgressViewSet(viewsets.ModelViewSet): queryset = Progress.objects.all() serializer_class =
ProgressSerializer class AdminViewSet(viewsets.ModelViewSet): queryset = Admin.objects.all()
serializer_class = AdminSerializer class ParentViewSet(viewsets.ModelViewSet): queryset =
Parent.objects.all() serializer_class = ParentSerializer class UserViewSet(viewsets.ModelViewSet):
queryset = User.objects.all() serializer_class = UserSerializer permission_classes =
[permissions.IsAuthenticated] @action(detail=False, methods=['post'],
permission_classes=[permissions.AllowAny]) def register(self, request): serializer =
UserRegistrationSerializer(data=request.data) if serializer.is_valid(): user = serializer.save() return
Response(UserSerializer(user).data, status=status.HTTP_201_CREATED) return
Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST) class
StudentViewSet(viewsets.ModelViewSet): queryset = Student.objects.all() serializer_class =
StudentSerializer permission_classes = [permissions.IsAuthenticated] @action(detail=True,
methods=['get']) def progress(self, request, pk=None): student = self.get_object() progress =
Progress.objects.filter(student=student) serializer = ProgressSerializer(progress, many=True)
return Response(serializer.data) @action(detail=True, methods=['get']) def badges(self, request,
pk=None): student = self.get_object() badges = student.badges.all() serializer =
BadgeSerializer(badges, many=True) return Response(serializer.data) @action(detail=True,
methods=['get']) def achievements(self, request, pk=None): """Récupère toutes les réalisations de
l'étudiant""" student = self.get_object() data = student.get_achievement_summary() return
Response(data) @action(detail=True, methods=['get']) def learning_path(self, request, pk=None):
"""Récupère le parcours d'apprentissage personnalisé""" student = self.get_object() data =
student.get_learning_path_status() return Response({ 'current_status': data,
'recommended_exercises': ExerciseSerializer( data['recommended_exercises'], many=True ).data })
@action(detail=True, methods=['get']) def subject_progress(self, request, pk=None): """Récupère
la progression par matière""" student = self.get_object() subject =
request.query_params.get('subject') if not subject: return Response( {'error': 'Le paramètre subject
est requis'}, status=status.HTTP_400_BAD_REQUEST ) data = student.get_subject_progress(subject)

```

```

return Response(data) class TeacherViewSet(viewsets.ModelViewSet): queryset =
Teacher.objects.all() serializer_class = TeacherSerializer permission_classes =
[permissions.IsAuthenticated] @action(detail=True, methods=['get']) def courses(self, request,
pk=None): teacher = self.get_object() courses = Course.objects.filter(teacher=teacher) serializer =
CourseSerializer(courses, many=True) return Response(serializer.data) @action(detail=True,
methods=['get']) def classes(self, request, pk=None): teacher = self.get_object() classes =
Class.objects.filter(teacher=teacher) serializer = ClassSerializer(classes, many=True) return
Response(serializer.data) class CourseViewSet(viewsets.ModelViewSet): queryset =
Course.objects.all() serializer_class = CourseSerializer permission_classes =
[permissions.IsAuthenticated] @action(detail=True, methods=['get']) def resources(self, request,
pk=None): course = self.get_object() resources = Resource.objects.filter(course=course) serializer
= ResourceSerializer(resources, many=True) return Response(serializer.data) @action(detail=True,
methods=['get']) def exercises(self, request, pk=None): course = self.get_object() exercises =
Exercise.objects.filter(course=course) serializer = ExerciseSerializer(exercises, many=True) return
Response(serializer.data) class ResourceViewSet(viewsets.ModelViewSet): queryset =
Resource.objects.all() serializer_class = ResourceSerializer permission_classes =
[permissions.IsAuthenticated] @action(detail=True, methods=['get']) def download(self, request,
pk=None): resource = self.get_object() if not resource.file: return Response( {'error': 'Aucun fichier
disponible'}, status=status.HTTP_404_NOT_FOUND ) resource.increment_download() response =
FileResponse(resource.file) response['Content-Disposition'] = f'attachment;
filename="{resource.file.name}" return response @action(detail=False, methods=['get']) def
offline_available(self, request): """Liste toutes les ressources disponibles hors ligne""" resources =
Resource.objects.filter(offline_available=True) serializer = self.get_serializer(resources, many=True)
return Response(serializer.data) @action(detail=True, methods=['get']) def metadata(self, request,
pk=None): """Récupère les métadonnées pour le stockage hors ligne""" resource =
self.get_object() data = { 'id': resource.id, 'title': resource.title, 'type': resource.type, 'file_size':
resource.file_size, 'last_modified': resource.modified_at, 'checksum': resource.get_checksum(), }
return Response(data) @action(detail=True, methods=['post']) def mark_offline(self, request,
pk=None): resource = self.get_object() resource.offline_available = True resource.save() return
Response({'message': 'Ressource marquée comme disponible hors ligne'}) class
ExerciseViewSet(viewsets.ModelViewSet): queryset = Exercise.objects.all() serializer_class =
ExerciseSerializer permission_classes = [permissions.IsAuthenticated] @action(detail=True,
methods=['post']) def submit_answer(self, request, pk=None): exercise = self.get_object() student
= request.user.student answers = request.data.get('answers', {}) progress =
exercise.submit_answer(student, answers) return Response({'score': progress.score, 'message':
'Réponse soumise avec succès' }) class ClassViewSet(viewsets.ModelViewSet): queryset =
Class.objects.all() serializer_class = ClassSerializer permission_classes =
[permissions.IsAuthenticated] @action(detail=True, methods=['post']) def add_student(self,
request, pk=None): class_obj = self.get_object() student_id = request.data.get('student_id')
student = get_object_or_404(Student, id=student_id) class_obj.add_student(student) return
Response({'message': 'Étudiant ajouté avec succès'}) class ExamViewSet(viewsets.ModelViewSet):
queryset = Exam.objects.all() serializer_class = ExamSerializer permission_classes =
[permissions.IsAuthenticated] def get_queryset(self): """Filtre les examens selon le rôle de
l'utilisateur""" user = self.request.user if hasattr(user, 'teacher'): return
Exam.objects.filter(course__teacher=user.teacher) elif hasattr(user, 'student'): return
Exam.objects.filter(course__classes__students=user.student) return Exam.objects.none()
@action(detail=True, methods=['post']) def start(self, request, pk=None): """Démarré une session
d'examen""" exam = self.get_object() student = request.user.student # Vérifie si l'étudiant peut
passer l'examen if not exam.course.classes.filter(students=student).exists(): return Response(
{'error': 'Vous n'êtes pas autorisé à passer cet examen'}, status=status.HTTP_403_FORBIDDEN ) #

```

```

Vérifie le nombre de tentatives attempts = Progress.objects.filter( student=student,
exercise_exams=exam ).count() if attempts >= exam.allowed_attempts: return Response( {'error':
'Nombre maximum de tentatives atteint'}, status=status.HTTP_400_BAD_REQUEST ) session =
exam.start_session(student) return Response({ 'session_id': session.id, 'remaining_time':
str(exam.duration), 'total_exercises': exam.exercises.count() }) @action(detail=True,
methods=['post']) def submit_answer(self, request, pk=None): """Soumet une réponse pendant
l'examen""" exam = self.get_object() exercise_id = request.data.get('exercise_id') answer =
request.data.get('answer') session_id = request.data.get('session_id') if not all([exercise_id, answer,
session_id]): return Response( {'error': 'Données manquantes'},
status=status.HTTP_400_BAD_REQUEST ) try: exercise = Exercise.objects.get(id=exercise_id,
exams=exam) session = ExamSession.objects.get(id=session_id) if session.is_time_up(): return
Response( {'error': 'Le temps est écoulé'}, status=status.HTTP_400_BAD_REQUEST )
session.submit_answer(exercise_id, answer) return Response({'message': 'Réponse enregistrée'})
except Exercise.DoesNotExist: return Response( {'error': 'Exercice non trouvé'},
status=status.HTTP_404_NOT_FOUND ) @action(detail=True, methods=['post']) def finish(self,
request, pk=None): """Termine l'examen et calcule le score""" exam = self.get_object() session_id
= request.data.get('session_id') try: session = ExamSession.objects.get(id=session_id) results =
exam.grade_exam(session.answers) # Enregistre les résultats for detail in results['details']:
Progress.objects.create( student=request.user.student, exercise=detail['exercise'],
score=detail['score'], completion_time=session.get_completion_time() ) # Génère le rapport report
= exam.generate_student_report(request.user.student) return Response({ 'score': results['score'],
'passed': results['passed'], 'report': report }) except ExamSession.DoesNotExist: return Response(
{'error': 'Session non trouvée'}, status=status.HTTP_404_NOT_FOUND ) @action(detail=True,
methods=['get']) def statistics(self, request, pk=None): """Récupère les statistiques de l'examen"""
exam = self.get_object() if not hasattr(request.user, 'teacher'): return Response( {'error': 'Accès non
autorisé'}, status=status.HTTP_403_FORBIDDEN ) stats = exam.get_statistics() return
Response(stats) @action(detail=True, methods=['post']) def schedule(self, request, pk=None):
"""Planifie l'examen pour une classe""" exam = self.get_object() class_id =
request.data.get('class_id') if not class_id: return Response( {'error': 'ID de classe requis'},
status=status.HTTP_400_BAD_REQUEST ) try: class_group = Class.objects.get(id=class_id)
exam.schedule_for_class(class_group) return Response({'message': 'Examen planifié avec succès'})
except Class.DoesNotExist: return Response( {'error': 'Classe non trouvée'},
status=status.HTTP_404_NOT_FOUND ) @action(detail=False, methods=['post']) def
create_practice(self, request): """Crée un examen blanc""" subject = request.data.get('subject')
difficulty = request.data.get('difficulty', 1) num_exercises = request.data.get('num_exercises', 10)
if not subject: return Response( {'error': 'Sujet requis'}, status=status.HTTP_400_BAD_REQUEST )
from .exam_management import create_practice_exam exam = create_practice_exam(subject,
difficulty, num_exercises) if not exam: return Response( {'error': 'Impossible de créer l'examen
blanc'}, status=status.HTTP_400_BAD_REQUEST ) serializer = self.get_serializer(exam) return
Response(serializer.data) @action(detail=True, methods=['post']) def submit(self, request,
pk=None): exam = self.get_object() student_answers = request.data.get('answers', {}) score =
exam.grade(student_answers) return Response({ 'score': score, 'message': 'Examen soumis avec
succès' }) class NotificationViewSet(viewsets.ModelViewSet): serializer_class =
NotificationSerializer permission_classes = [permissions.IsAuthenticated] def get_queryset(self):
return Notification.objects.filter(recipient=self.request.user) @action(detail=True,
methods=['post']) def mark_as_read(self, request, pk=None): notification = self.get_object()
notification.mark_as_read() return Response({'message': 'Notification marquée comme lue'}) class
BadgeViewSet(viewsets.ModelViewSet): queryset = Badge.objects.all() serializer_class =
BadgeSerializer permission_classes = [permissions.IsAuthenticated] @action(detail=False,
methods=['get']) def available(self, request): """Liste les badges disponibles pour l'étudiant"""

```

```

student = request.user.student badges = Badge.objects.exclude(students=student) serializer =
self.get_serializer(badges, many=True) return Response(serializer.data) @action(detail=False,
methods=['get']) def earned(self, request): """Liste les badges gagnés par l'étudiant""" student =
request.user.student badges = student.badges.all() serializer = self.get_serializer(badges,
many=True) return Response(serializer.data) class LicenseViewSet(viewsets.ModelViewSet):
serializer_class = LicenseSerializer permission_classes = [permissions.IsAuthenticated] def
get_queryset(self): user = self.request.user if hasattr(user, 'admin'): return License.objects.all()
return License.objects.filter(user=user) @action(detail=True, methods=['post']) def renew(self,
request, pk=None): """Renouvelle une licence""" license = self.get_object() duration =
request.data.get('duration', 1) try: license.renew(duration_months=duration) return Response({
'message': 'Licence renouvelée avec succès', 'license': LicenseSerializer(license).data }) except
ValidationError as e: return Response( {'error': str(e)}, status=status.HTTP_400_BAD_REQUEST )
class PaymentViewSet(viewsets.ModelViewSet): serializer_class = PaymentSerializer
permission_classes = [permissions.IsAuthenticated] def get_queryset(self): user = self.request.user
if hasattr(user, 'admin'): return Payment.objects.all() return Payment.objects.filter(user=user)
@action(detail=True, methods=['post']) def process(self, request, pk=None): """Traite un
paiement""" payment = self.get_object() if payment.status != 'pending': return Response( {'error':
'Ce paiement ne peut plus être traité'}, status=status.HTTP_400_BAD_REQUEST ) success =
payment.process_payment() if success: return Response({ 'message': 'Paiement traité avec succès',
'payment': PaymentSerializer(payment).data }) return Response( {'error': 'Le paiement a échoué'},
status=status.HTTP_400_BAD_REQUEST ) @action(detail=True, methods=['get']) def receipt(self,
request, pk=None): """Génère un reçu de paiement""" payment = self.get_object() if
payment.status != 'completed': return Response( {'error': 'Aucun reçu disponible'},
status=status.HTTP_400_BAD_REQUEST ) receipt = payment.generate_receipt() return
Response(receipt) class SubscriptionViewSet(viewsets.ModelViewSet): serializer_class =
SubscriptionSerializer permission_classes = [permissions.IsAuthenticated] def get_queryset(self):
user = self.request.user if hasattr(user, 'admin'): return Subscription.objects.all() return
Subscription.objects.filter(user=user) @action(detail=True, methods=['post']) def
toggle_auto_renew(self, request, pk=None): """Active/désactive le renouvellement automatique"""
subscription = self.get_object() subscription.auto_renew = not subscription.auto_renew
subscription.save() return Response({ 'message': 'Préférence de renouvellement mise à jour',
'subscription': SubscriptionSerializer(subscription).data }) @action(detail=True, methods=['post'])
def change_interval(self, request, pk=None): """Change l'intervalle de facturation""" subscription =
self.get_object() new_interval = request.data.get('interval') if new_interval not in
dict(Subscription.INTERVAL_CHOICES): return Response( {'error': 'Intervalle invalide'},
status=status.HTTP_400_BAD_REQUEST ) subscription.interval = new_interval
subscription.update_next_billing_date() return Response({ 'message': 'Intervalle de facturation mis
à jour', 'subscription': SubscriptionSerializer(subscription).data }) class
MobilePaymentCallbackView(APIView): """Gère les callbacks des fournisseurs de paiement
mobile""" permission_classes = [] # Pas d'authentification requise def post(self, request, provider):
reference = request.data.get('reference') if not reference: return Response( {'error': 'Référence
manquante'}, status=status.HTTP_400_BAD_REQUEST ) try: mobile_payment =
MobilePayment.objects.get( provider=provider, reference=reference ) # Traite le callback
new_status = mobile_payment.process_callback(request.data) return Response({ 'status':
new_status, 'message': 'Callback traité avec succès' }) except MobilePayment.DoesNotExist: return
Response( {'error': 'Paiement non trouvé'}, status=status.HTTP_404_NOT_FOUND )

```

Votre code pour les vues Django REST Framework (DRF) est bien structuré et couvre une grande variété de fonctionnalités, allant de la gestion des utilisateurs à la gestion des paiements, en

passant par les examens et les notifications. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```
from django.urls import path, include from rest_framework.routers import DefaultRouter from
.views import ParentViewSet, AdminViewSet from rest_framework_simplejwt.views import (
TokenObtainPairView, TokenRefreshView, ) from .views import ( UserViewSet, StudentViewSet,
TeacherViewSet, ParentViewSet, AdminViewSet, CourseViewSet, ResourceViewSet,
ExerciseViewSet, ProgressViewSet, ClassViewSet, BadgeViewSet, NotificationViewSet,
ExamViewSet, MobilePaymentViewSet, MobilePaymentConfigViewSet,
MobilePaymentCallbackView ) router = DefaultRouter() router.register(r'users', UserViewSet)
router.register(r'students', StudentViewSet) router.register(r'teachers', TeacherViewSet)
router.register(r'parents', ParentViewSet) router.register(r'admins', AdminViewSet)
router.register(r'courses', CourseViewSet) router.register(r'resources', ResourceViewSet)
router.register(r'exercises', ExerciseViewSet) router.register(r'progress', ProgressViewSet)
router.register(r'classes', ClassViewSet) router.register(r'badges', BadgeViewSet)
router.register(r'notifications', NotificationViewSet, basename='notification')
router.register(r'exams', ExamViewSet) # router.register(r'mobile-payments',
MobilePaymentViewSet) router.register(r'payment-configs', MobilePaymentConfigViewSet)
urlpatterns = [ path("", include(router.urls)), path('token/', TokenObtainPairView.as_view(),
name='token_obtain_pair'), path('token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'), # URLs pour les callbacks de paiement mobile path( 'payment-
callback/<str:provider>', MobilePaymentCallbackView.as_view(), name='mobile-payment-
callback' ), ]
```

The server is busy. Please try again later.

```
from django.urls import path, include from rest_framework.routers import DefaultRouter from
.views import ParentViewSet, AdminViewSet from rest_framework_simplejwt.views import (
TokenObtainPairView, TokenRefreshView, ) from .views import ( UserViewSet, StudentViewSet,
TeacherViewSet, ParentViewSet, AdminViewSet, CourseViewSet, ResourceViewSet,
ExerciseViewSet, ProgressViewSet, ClassViewSet, BadgeViewSet, NotificationViewSet,
ExamViewSet, MobilePaymentViewSet, MobilePaymentConfigViewSet,
MobilePaymentCallbackView ) router = DefaultRouter() router.register(r'users', UserViewSet)
router.register(r'students', StudentViewSet) router.register(r'teachers', TeacherViewSet)
router.register(r'parents', ParentViewSet) router.register(r'admins', AdminViewSet)
router.register(r'courses', CourseViewSet) router.register(r'resources', ResourceViewSet)
router.register(r'exercises', ExerciseViewSet) router.register(r'progress', ProgressViewSet)
router.register(r'classes', ClassViewSet) router.register(r'badges', BadgeViewSet)
router.register(r'notifications', NotificationViewSet, basename='notification')
router.register(r'exams', ExamViewSet) # router.register(r'mobile-payments',
MobilePaymentViewSet) router.register(r'payment-configs', MobilePaymentConfigViewSet)
urlpatterns = [ path("", include(router.urls)), path('token/', TokenObtainPairView.as_view(),
name='token_obtain_pair'), path('token/refresh/', TokenRefreshView.as_view(),
name='token_refresh'), # URLs pour les callbacks de paiement mobile path( 'payment-
callback/<str:provider>', MobilePaymentCallbackView.as_view(), name='mobile-payment-
callback' ), ]
```

Votre fichier `urls.py` est bien structuré et utilise le routeur `DefaultRouter` de Django REST Framework (DRF) pour générer automatiquement les URLs pour vos vues basées sur les `ViewSets`. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```
from django.db.models.signals import post_save from django.dispatch import receiver from
.models import Progress, Exercise, Student from datetime import timedelta from django.utils
import timezone # Points bonus pour différentes actions STREAK_BONUS = { 7: 100, # Bonus pour
7 jours consécutifs 30: 500, # Bonus pour 30 jours consécutifs 90: 1500, # Bonus pour 90 jours
consécutifs } DIFFICULTY_MULTIPLIER = { 1: 1.0, # Facile 2: 1.2, # Moyen 3: 1.5, # Difficile 4: 2.0, #
Expert } SPEED_BONUS_THRESHOLD = 0.7 # 70% du temps moyen def
calculate_exercise_points(exercise, score, completion_time=None): """Calcule les points pour un
exercice en fonction de la difficulté et du temps""" base_points = exercise.points difficulty_bonus
= DIFFICULTY_MULTIPLIER.get(exercise.difficulty, 1.0) points = base_points * difficulty_bonus *
(score / 100) # Bonus de vitesse si applicable if completion_time and
exercise.average_completion_time: if completion_time < exercise.average_completion_time *
SPEED_BONUS_THRESHOLD: points *= 1.2 # 20% bonus pour rapidité return int(points) def
check_streak_bonus(student): """Vérifie et attribue les bonus de série""" for days, bonus in
STREAK_BONUS.items(): if student.streak_days == days: student.add_points(bonus) return True
return False @receiver(post_save, sender=Progress) def handle_exercise_completion(sender,
instance, created, **kwargs): """Gère les événements après la complétion d'un exercice""" if not
created: return student = instance.student exercise = instance.exercise # Met à jour le compteur
d'exercices student.total_exercises_completed += 1 # Calcule et ajoute les points points =
calculate_exercise_points( exercise, instance.score, instance.completion_time )
student.add_points(points) # Met à jour la série d'activités student.update_streak() # Vérifie les
bonus de série check_streak_bonus(student) student.save() def
award_special_achievement(student, achievement_type, points=0): """Attribue des réalisations
spéciales""" from .models import Badge SPECIAL_ACHIEVEMENTS = { 'first_perfect_score': {
'name': 'Premier 100%', 'description': 'Premier exercice parfait', 'points': 100 }, 'subject_mastery': {
'name': 'Maître du sujet', 'description': 'Excellence dans une matière', 'points': 500 },
'speed_demon': { 'name': 'Éclair', 'description': 'Completion rapide avec précision', 'points': 200 } }
if achievement_type in SPECIAL_ACHIEVEMENTS: achievement =
SPECIAL_ACHIEVEMENTS[achievement_type] badge = Badge.objects.create(
name=achievement['name'], description=achievement['description'], required_points=0 # Badge
spécial ) badge.award(student) student.add_points(achievement['points'] + points)
```

Votre code pour la gestion des signaux et des points dans votre application est bien structuré et couvre plusieurs aspects importants, comme le calcul des points, la gestion des séries d'activités, et l'attribution de badges spéciaux. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```
# Register your models here. # keysGen/admin.py from django.contrib import admin from
.models import License @admin.register(License) class ActivationKeyAdmin(admin.ModelAdmin):
list_display = ('key', 'created_at', 'is_used')
```

Votre fichier `admin.py` pour l'application `keysGen` est bien structuré et enregistre le modèle `License` dans l'interface d'administration de Django. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```
from django.db import models import uuid # Create your models here. class
License(models.Model): key = models.UUIDField(default=uuid.uuid4, editable=False,
unique=True) created_at = models.DateTimeField(auto_now_add=True) is_used =
models.BooleanField(default=False) def __str__(self): return str(self.key) def generate_key(cls):
new_key = cls() new_key.save() return new_key.key
```

Votre modèle `License` est bien structuré et utilise un `UUIDField` pour générer des clés de licence uniques. Voici une analyse détaillée de votre code et des suggestions pour l'améliorer.

```
# keysGen/urls.py from django.urls import path from .views import generate_activation_key
urlpatterns = [ path('generate-key/', generate_activation_key, name='generate_activation_key'), ]
```

Time flies when we're having a great chat! We've exceeded the length limit. Please start a new chat!

```
from django.shortcuts import render from .models import License from django.http import
JsonResponse # Create your views here. def generate_activation_key(request): if request.method
== 'POST': key = License.generate_key() return JsonResponse({'status': 'success', 'key': str(key)})
return JsonResponse({'status': 'error', 'message': 'Méthode non autorisée'})
```

Time flies when we're having a great chat! We've exceeded the length limit. Please start a new chat!

```
from django.shortcuts import render from .models import License from django.http import
JsonResponse # Create your views here. def generate_activation_key(request): if request.method
== 'POST': key = License.generate_key() return JsonResponse({'status': 'success', 'key': str(key)})
return JsonResponse({'status': 'error', 'message': 'Méthode non autorisée'})
```