Pour implémenter les fonctionnalités demandées (Simplification des Modèles 3D, Conversion de Labels, et Coloration et Attributs) en utilisant Django, voici un guide détaillé étape par étape pour chaque fonctionnalité. Django est un framework web puissant qui te permettra de gérer facilement le backend de ton application.

1. Configuration initiale avec Django

1. Installer Django :
   - Assure-toi d'avoir Python installé sur ta machine.
   - Installe Django via pip :

```
pip install django
```

2. Créer un nouveau projet Django :
   - Crée un nouveau projet Django :

```
django-admin startproject ton_projet
```

   - Crée une nouvelle application au sein du projet pour gérer les fonctionnalités :

```
python manage.py startapp models3d
```

3. Configurer le projet :
   - Ajoute ton application (`models3d`) au fichier `settings.py` sous `INSTALLED_APPS` .
   - Configure la base de données dans `settings.py` si nécessaire (par défaut, Django utilise SQLite).
4. Démarrer le serveur :
   - Démarre le serveur de développement pour vérifier que tout fonctionne :

```
python manage.py runserver
```

2. Simplification des Modèles 3D

La simplification de modèles 3D consiste à réduire le nombre de polygones pour rendre le modèle plus léger. Pour cela, tu peux utiliser des bibliothèques comme `PyMesh` ou `Trimesh`.

Étapes :

1. Installer les bibliothèques nécessaires :

```
pip install pymesh trimesh
```

2. Créer une vue pour le traitement des fichiers 3D :
   - Dans ton application `models3d`, crée une vue qui recevra les fichiers 3D via une requête HTTP POST.
   - Le fichier est ensuite traité pour être simplifié.

```
# models3d/views.py
import trimesh
from django.http import JsonResponse
from django.views import View
from django.core.files.storage import default_storage

class SimplifyModelView(View):
    def post(self, request):
        uploaded_file = request.FILES['file']
        file_path = default_storage.save(uploaded_file.name, uploaded_file)

        mesh = trimesh.load(file_path)
        simplified_mesh = mesh.simplify_quadratic_decimation(target_faces=1000)  # Ex : Réduire à 1000 faces

        simplified_file_path = f'simplified_{uploaded_file.name}'
        simplified_mesh.export(simplified_file_path)

        return JsonResponse({'simplified_model_url': simplified_file_path})
```

3. Configurer l'URL pour cette vue :
   - Dans `models3d/urls.py`, configure l'URL :

```
from django.urls import path
from .views import SimplifyModelView

```

```
urlpatterns = [
    path('simplify/', SimplifyModelView.as_view(), name='simplify_model'),
]
```

4. Intégration dans Next.js :
    - Depuis le frontend, tu envoies une requête POST avec le modèle 3D vers l'URL `/simplify/` de ton serveur Django.

3. Conversion de Labels

La conversion de labels peut impliquer de changer le format des annotations ou de les mapper à d'autres valeurs selon les besoins.

Étapes :

1. Définir une structure de données pour les labels :
    - Crée un modèle Django pour stocker les labels :

```
# models3d/models.py
from django.db import models

class Label(models.Model):
    name = models.CharField(max_length=255)
    data = models.JSONField()

    def __str__(self):
        return self.name
```

2. Créer une vue pour la conversion des labels :
    - Crée une vue qui prendra des labels en entrée et les convertira selon un format spécifique.

```
# models3d/views.py
class ConvertLabelView(View):
    def post(self, request):
        labels = request.POST.get('labels')
        # Supposons que les labels soient au format JSON
        converted_labels = self.convert_labels(labels)
        return JsonResponse({'converted_labels': converted_labels})

    def convert_labels(self, labels):
        # Implémenter la logique de conversion des labels ici
        converted = labels.upper()  # Exemple simple de conversion
```

```
        return converted
```

3. Configurer l'URL pour cette vue :

```
from .views import ConvertLabelView

urlpatterns += [
    path('convert-labels/', ConvertLabelView.as_view(), name='convert_labels'),
]
```

4. Intégration dans Next.js :
   - Le frontend peut envoyer les labels via une requête POST à l'URL `/convert-labels/`.

4. Coloration et Attributs

La gestion de la coloration et des attributs implique de modifier des propriétés des modèles 3D, comme la couleur des surfaces, les textures, etc.

Étapes :

1. Créer une vue pour la gestion des couleurs et des attributs :
   - Crée une vue qui recevra les instructions de modification de la couleur et des attributs des modèles 3D.

```
# models3d/views.py
class UpdateAttributesView(View):
    def post(self, request):
        model_id = request.POST.get('model_id')
        color = request.POST.get('color')
        # Charger le modèle 3D correspondant et appliquer les changements
        mesh = trimesh.load_model(model_id)
        mesh.visual.vertex_colors = color  # Appliquer la couleur

        # Sauvegarder les changements
        updated_file_path = f'colored_{model_id}.obj'
        mesh.export(updated_file_path)

        return JsonResponse({'updated_model_url': updated_file_path})
```

2. Configurer l'URL pour cette vue :

```
from .views import UpdateAttributesView

urlpatterns += [
    path('update-attributes/', UpdateAttributesView.as_view(), name='update_attributes'),
]
```

3. Intégration dans Next.js :
    - Le frontend envoie les instructions de mise à jour des attributs via une requête POST à `/update-attributes/`.

Conclusion

1. Développement : Tu configures Django pour gérer les requêtes et le traitement des modèles 3D, des labels, et des attributs.
2. API Routes : Chaque fonctionnalité (simplification, conversion, coloration) est accessible via une route API spécifique.
3. Intégration : Tu intègres ces fonctionnalités avec ton frontend Next.js en utilisant des requêtes HTTP.

En suivant ce processus, tu auras un backend Django capable de gérer les fonctionnalités requises pour ton projet.