DRF Generic Views

Generic Views enthalten vorgefertigte Logik für Standard-Operationen

Definition

Häufig verwendete Views, die sich eng an Datenbankmodelle anlehnen (z. B. Erstellen einer Modellinstanz, Löschen einer Instanz, Auflisten von Instanzen usw.) sind in Django REST Framework Views (DRF) bereits vordefiniert.

Diese Teile des wiederverwendbaren Verhaltens werden als Generic Views bezeichnet.

Die generische Basisklasse GenericAPIView

Alle generischen Views leiten sich von der GenericAPIView ab.

Diese erweitert die Funktionalität der grundlegenden APIView Klasse, indem sie zusätzliche, häufig benötigte Verhaltensweisen für Standard-Listen- und Detailansichten hinzufügt.

Diese Erweiterungen machen die GenericAPIView besonders nützlich für den Umgang mit typischen Datenbankabfragen und -operationen.

Mixins

Verwendung von Mixins mit GenericAPIView:

Die konkreten generischen Views in DRF, wie ListAPIView, CreateAPIView, RetrieveAPIView, UpdateAPIView, und DestroyAPIView, sind Kombinationen aus GenericAPIView und einem oder mehreren Mixin-Klassen. Diese Mixins fügen spezifische Verhaltensweisen für verschiedene CRUD-Operationen hinzu:

ListModelMixin: Fügt Methoden hinzu, um eine Liste von Objekten zu erhalten.

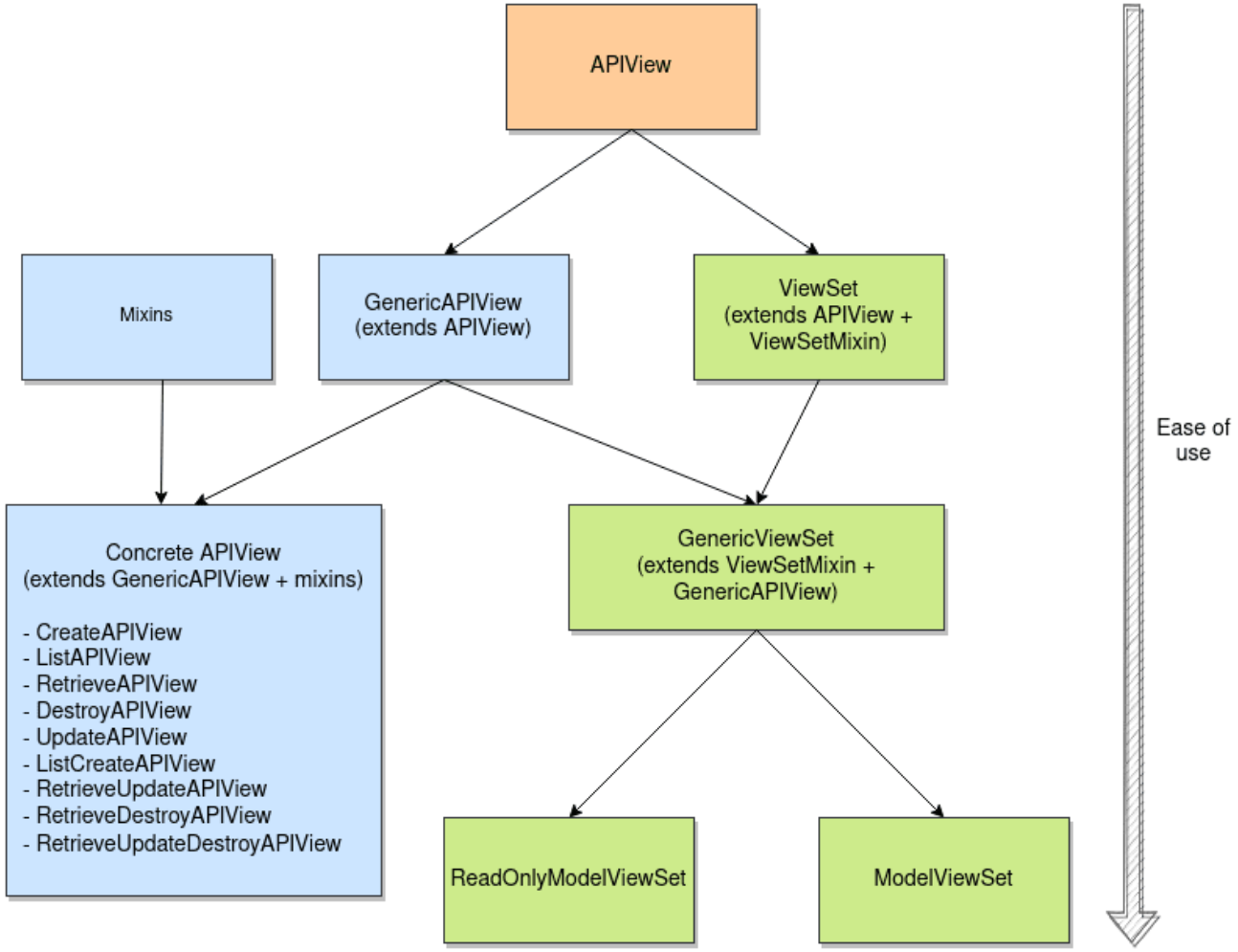
CreateModelMixin: Ermöglicht das Erstellen eines neuen Objekts.

RetrieveModelMixin: Dient dem Abrufen eines einzelnen Objekts.

UpdateModelMixin: Ermöglicht das Aktualisieren eines Objekts.

DestroyModelMixin: Stellt Methoden zum Löschen eines Objekts bereit.

Aufbau der generischen Views



Views für eine Model Instanz

- RetrieveAPIView: ein Objekt anfordern
- CreateAPIView: ein Objekt anlegen
- DestroyAPIView: ein Objekt löschen
- UpdateAPIView: ein Objekt updaten
- RetrieveUpdateAPIView: ein Objekt anfordern und updaten
- RetrieveDestroyAPIView: ein Objekt anfordern und löschen
- RetrieveUpdateDestroyAPIView: Anfordern, updaten und löschen

Views für Listen von Objekten

ListAPIView: Liste von Objekten anfordern

ListCreateAPIView: Liste von Objekten und Anlegen eines Objekts

allg. Methoden der generischen Views: get_serializer_class

get_serializer_class(self) -> serializers.Serializer
Gebe einen Serializer zurück, zb. in Abhängigkeit von der Methode

```
def get_serializer_class(self):
    if self.request.method == "POST":
        return InsertItemSerializer
    return ItemSerializer
```

Defaultwert ist der Serializer, der in dem serializer_class-Attribut der View festgelegt wurde. Nützlich, wenn sich zum Beispiel der Serializer für das Anlegen und Auflisten in einer ListCreateAPIView unterscheidet.

allg. Methoden der generischen Views: get_queryset

Diese Methode wird verwendet, um die Datenquelle zu bestimmen. Sie können diese Methode überschreiben, um eine benutzerdefinierte QuerySet-Logik bereitzustellen. Nützlich, wenn zum Beispiel per prefetch_related Daten vorgeladen werden oder Daten gefiltert werden müssen.

```
def get_queryset(self) -> Queryset:
        return Event.objects.filter(is_active=True)
```

Wichtige Methoden spezifischer generischer Views

1) ListAPIView (Objekte auflisten, GET-Methode)
2) CreateAPIView (Objekte anlegen, POST-Methode)
3) RetrieveAPIView (Objekt anfordern, GET-Methode)
4) DestroyAPIView (Objekt löschen, DELETE-Methode)
5) UpdateAPIView (Objekt update, PATCH / PUT-Methode)

die urlpatterns für die Beispiele sehen so aus:

```
urlpatterns = [
    path("items/", ItemListView.as_view(), name="item-list"),
    path("items/create/", ItemCreateView.as_view(), name="item-create"),
    path("items/<int:pk>/", ItemDetailView.as_view(), name="item-detail"),
    path("items/<int:pk>/update/", ItemUpdateView.as_view(), name="item-update"),
    path("items/<int:pk>/delete/", ItemDestroyView.as_view(), name="item-delete"),
]
```

1) wichtige Methoden der ListView

Die ListView gibt eine Liste von Objekten zurück. Hier sind einige wichtige Methoden.

list(self, request, *args, **kwargs):

Diese ist die Hauptmethode, die für das Abrufen und Senden der Liste der Objekte zuständig ist. Sie wird aufgerufen, wenn eine GET-Anfrage an den Endpunkt gesendet wird. Sie wird selten überschrieben.

filter_queryset(self, queryset):

Hier können Sie die Ergebnisse weiter filtern, nachdem Sie die ursprüngliche QuerySet von get_queryset erhalten haben. Diese Methode wird häufig mit Filter-Backends verwendet, um dynamisches Filtern zu ermöglichen. Einfaches Filtern anhand der Query-Parameter kann auch in der Methode get_queryset durchgeführt werden.

Beispiel einer ListAPIView

```
class ItemListView(generics.ListAPIView):
    serializer_class = ItemSerializer

    def get_queryset(self):
        queryset = Item.objects.all()
        # Erhalten Sie die Filterparameter aus der Anfrage
        name = self.request.query_params.get('name')

        # Filtern Sie das QuerySet basierend auf den Parametern
        if name:
            queryset = queryset.filter(name__icontains=name)

        return queryset
```

Eine URL könnte so aussehen: https://example.com/api/items?name="Bob"

2) wichtige Methoden der CreateAPIVIEW

create(self, request, *args, **kwargs):

Diese Methode wird aufgerufen, um eine neue Ressource zu erstellen. Sie nimmt die eingehenden Daten aus der request, deserialisiert und validiert sie mit dem zugehörigen Serializer.

Bei erfolgreicher Validierung speichert sie die Daten und gibt eine HTTP 201-Antwort zurück, im Falle eines Validierungsfehlers eine HTTP 400-Antwort.

perform_create(self, serializer):

Diese Methode wird von create() aufgerufen, nachdem der Serializer die Daten validiert hat.

Die Standardimplementierung ruft einfach save() auf dem Serializer auf, aber diese Methode kann überschrieben werden, um zusätzliche Logik auszuführen, nachdem die Instanz gespeichert wurde, z. B. das Senden eines Signals oder das Ausführen einer benutzerdefinierten Aktion.

```
class ItemCreateView(CreateAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer

    def perform_create(self, serializer):
        # Hier können Sie benutzerdefinierte Logik hinzufügen, bevor das Objekt gespeichert wird.
        # Zum Beispiel:
        # Prüfen Sie, ob der Benutzer berechtigt ist, ein Item zu erstellen.
        # Führen Sie zusätzliche Validierungen durch.
        # Setzen Sie den Besitzer eines Objekts mit dem eingeloggten User
        user = self.request.user
        # etc.
        serializer.save(author=user)  # ruft intern serializer.create() auf
```

3. Wichtige Methoden der RetrieveAPIView

Die RetrieveAPIView ist speziell für das Abrufen eines einzelnen Objekts aus der Datenbank vorgesehen.

retrieve(self, request, *args, **kwargs):

Dies ist die Hauptmethode, die aufgerufen wird, wenn eine GET-Anfrage an die View gesendet wird. Wird in der Regel nicht überschrieben.

get_object(self):

Diese Methode wird verwendet, um das spezifische Objekt basierend auf der übergebenen URL zu holen. Kann überschrieben werden, wenn der Lookup komplexer ist, zb. bei mehreren Attributen.

Beispiel einer RetrieveAPIView mit komplexem Lookup

```
class ItemDetailView(generics.RetrieveAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer

    def get_object(self) -> Item:
        # Retrieve the URL parameters
        category = self.kwargs.get('category')
        id = self.kwargs.get('id')

        # Perform the lookup filtering by both `category` and `id`
        try:
            return self.queryset.get(category=category, id=id)
        except Item.DoesNotExist:
            raise NotFound(detail="Item not found with category={} and id={}".format(category, id))
```

4. Wichtige Methoden der DestroyAPIView

Die DestroyAPIView ist speziell für das Löschen von Objekten konzipiert. Sie bietet eine einfache Möglichkeit, DELETE-Anfragen in Ihrer API zu verarbeiten. Hier sind einige der wichtigsten Methoden, die Sie in einer DestroyAPIView verwenden können:

destroy(self, request, *args, **kwargs):

Die destroy-Methode wird aufgerufen, wenn eine DELETE-Anfrage an die View gesendet wird. Sie ruft get_object auf, um das zu löschende Objekt zu erhalten, und löscht es dann. Anschließend gibt sie eine entsprechende Antwort zurück.

delete(self, request, *args, **kwargs):

Diese Methode wird von der destroy-Methode aufgerufen und ist verantwortlich für die Verarbeitung der DELETE-Anfrage. Sie können diese Methode überschreiben, um benutzerdefinierte Löschlogiken oder Antwortverhalten zu implementieren.

In der Regel müssen selten Methoden der DestroyAPIView überschrieben werden.

Beispiel einer DestroyAPIView

```
class ItemDestroyView(generics.DestroyAPIView):
    queryset = Item.objects.all()

    def delete(self, request, *args, **kwargs):
        # Optional: Fügen Sie hier benutzerdefinierte Logik vor dem Löschen ein
        return super().delete(request, *args, **kwargs)

    def destroy(self, request, *args, **kwargs):
        instance = self.get_object()
        self.perform_destroy(instance)
        return Response(status=status.HTTP_204_NO_CONTENT)

    def perform_destroy(self, instance):
        instance.delete()
```

5. wichtige Methoden der UpdateAPIView

Die UpdateAPIView ist speziell für das Aktualisieren von Objekten konzipiert. Sie stellt eine Reihe von Methoden zur Verfügung, die das Aktualisieren von Daten in Ihrer API vereinfachen.

update(self, request, *args, **kwargs):

Die update-Methode wird aufgerufen, wenn eine PUT- oder PATCH-Anfrage an die View gesendet wird. Sie ist verantwortlich für das Abrufen des Objekts, das Aktualisieren seiner Daten basierend auf der Anfrage und das Zurückgeben der aktualisierten Daten.

put(self, request, *args, **kwargs):

Diese Methode behandelt PUT-Anfragen. Sie ruft in der Regel update auf, mit partial=False, was bedeutet, dass das gesamte Objekt aktualisiert wird.

patch(self, request, *args, **kwargs):

Diese Methode behandelt PATCH-Anfragen. Im Gegensatz zu put wird hier update mit partial=True aufgerufen, was bedeutet, dass nur die in der Anfrage bereitgestellten Felder aktualisiert werden.

Beispiel einer einfachen UpdateAPIView

```
class ItemUpdateView(generics.UpdateAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer
```

Zusammenfassen von Endpunkten

Die ListCreateAPIView, RetrieveUpdateAPIView, RetrieveDestroyAPIView und RetrieveUpdateDestroyAPIView fassen Endpunkte in einer View zusammen. Folglich lässt sich auch das urlpatetterns vereinfachen:

```
urlpatterns = [
    path("items/", ItemListCreateView.as_view(), name="item-list-create"),
    path(
        "items/<int:pk>/",
        ItemRetrieveUpdateDestroyView.as_view(),
        name="item-retrieve-update-destroy",
    ),
]
```