Django Restframework

ein leistungsstarkes und flexibles Toolkit für die Erstellung von Web-APIs

Features

HTTP Response Handling

Validierung

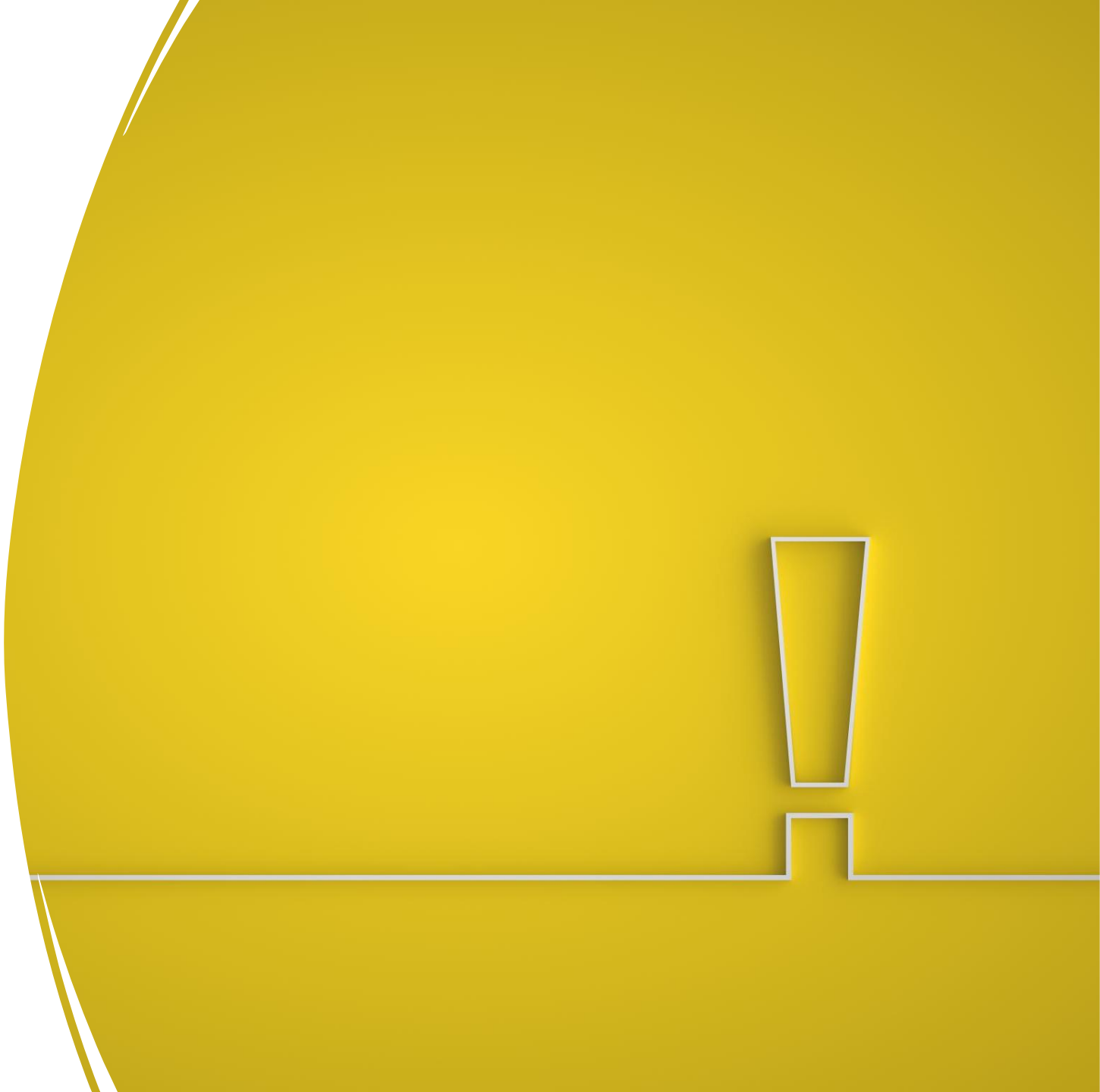
Pagination

Caching

Serialization

Rechte (Permissions)

Authetifizierungsmöglichkeiten (zb. per Token)



Warum Django RestFramework?

- es unterstützt klassenbasierte Views
- es orientiert sich stark an bekannte Django-Mechanismen, wie zum Beispiel Forms.
- es ist straight forward einzurichten
- ausgezeichnete Doku
- große Community

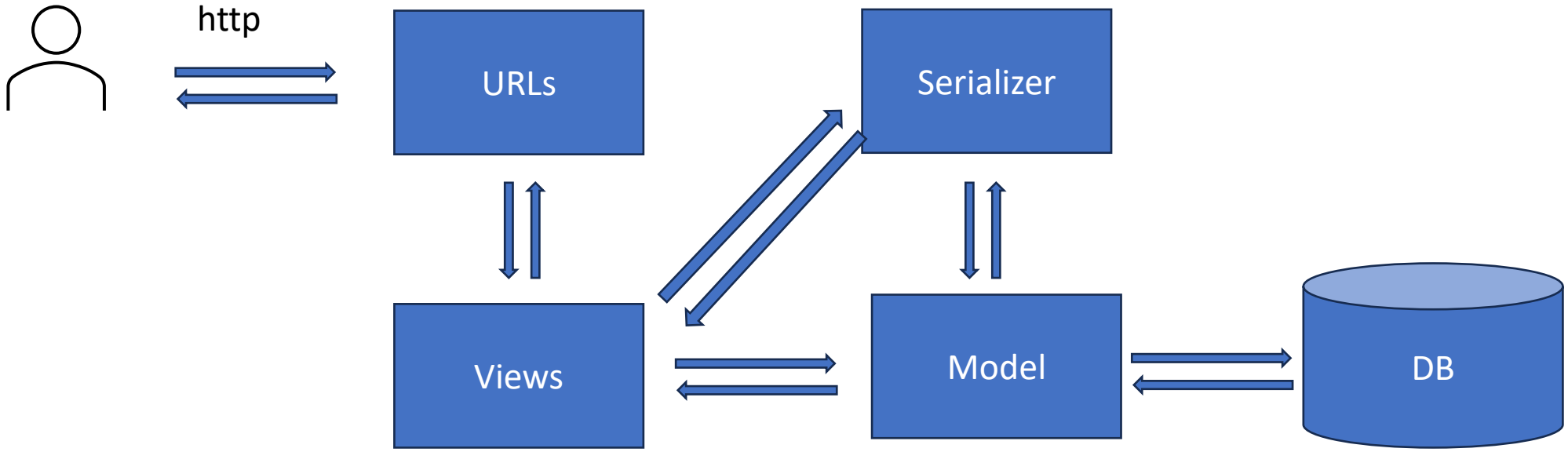
Key Elements

Serializers: deserialisieren eingehende und serialisieren ausgehende Daten. Validierung der eingehenden Daten.

Views: verarbeiten der User-Request

URLs: Routes zu den Endpunkten

Model View Serializer



Was ist ein Serializer?

Ein Serializer in Django Rest Framework ist ein Werkzeug, das komplexe Datentypen wie Querysets und Modelinstanzen in JSON umgewandelt werden können. Dieser Prozess wird auch als "Serialisierung" bezeichnet.

Ebenso kann der Serializer verwendet werden, um JSON-Daten in komplexe Datentypen umzuwandeln, was als "Deserialisierung" bekannt ist.

Somit spielt der Serializer eine zentrale Rolle in RESTful APIs, indem er die Kommunikation zwischen dem Datenmodell einer Anwendung und dem Format ermöglicht, das für die API-Endpunkte benötigt wird.

einen Serializer von einem Model ableiten

In den meisten Fällen will man ein Model serialisieren. Dafür kann man einen Serializer von einem Model ableiten. Das funktioniert ähnlich wie bei den Django ModelForms.

```
class PostSerializer(serializers.ModelSerializer):
    class Meta:
        model = Post
```

Beispiel eines Serializers

ein einfaches Model in models.py und ein einfacher Serializer in serializers.py

```
class Post(models.Model):
    name = models.CharField(max_length=20)
    content = models.TextField()

class PostSerializer(serializers.ModelSerializer):
     class Meta:
         model = Post
```

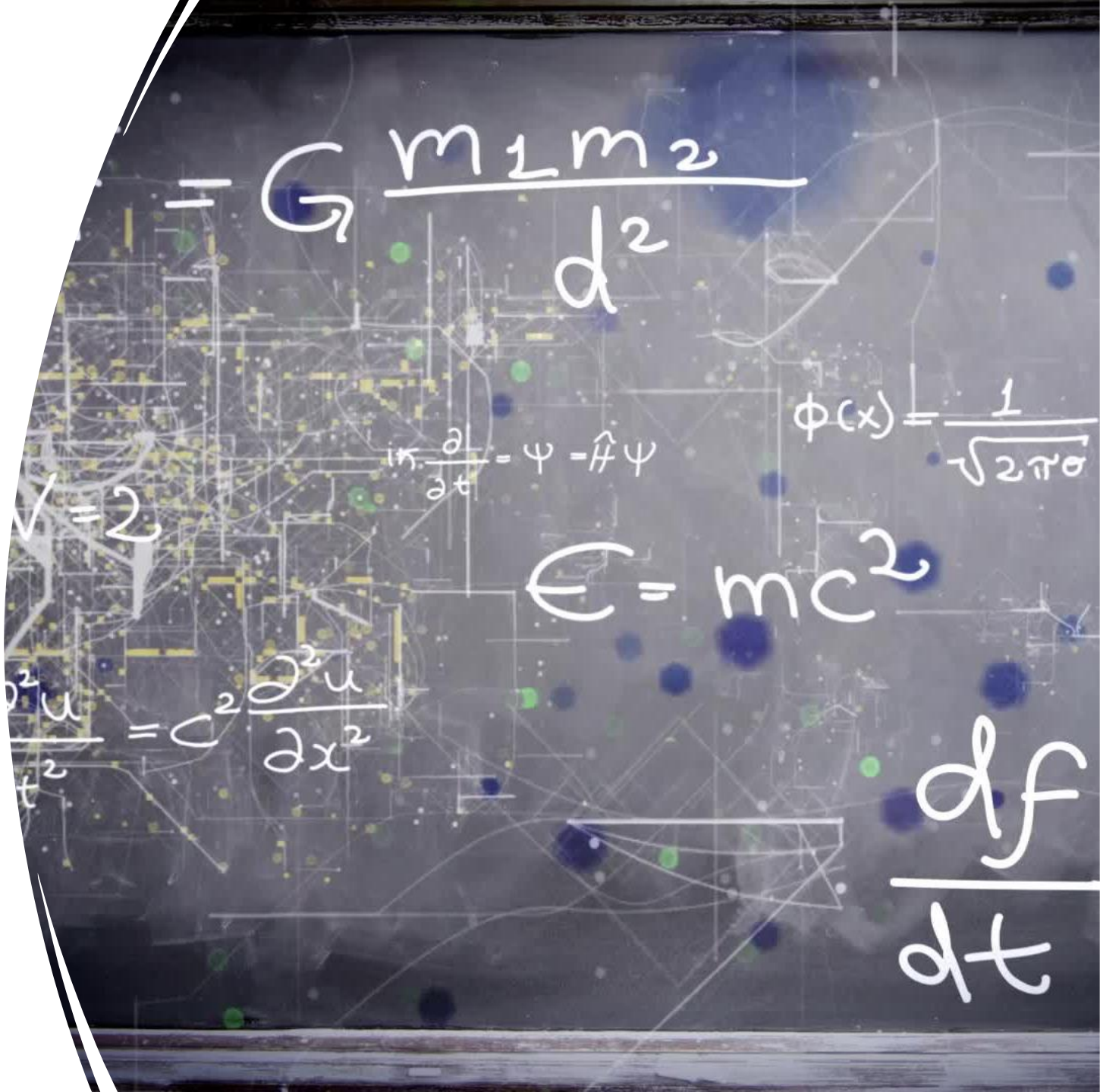
model-unabhängiger Serializer

Falls man den Serializer nicht von einem Model aufbauen will, kann man auch eigene Felder im Serializer angeben

```
class EmailSerializer(serializers.Serializer):

    email = serializers.EmailField()

    content = serializers.CharField(max_length=500)
```



Was ist eine View?

In Django Rest Framework bezeichnet eine "View" eine Komponente, die die Logik zur Verarbeitung von HTTP-Anfragen steuert. Sie nimmt Anfragen entgegen, führt die notwendigen Aktionen aus (wie das Abrufen von Daten aus der Datenbank oder das Bearbeiten von Daten), und gibt eine Antwort zurück.

Im Kontext von Web-APIs sind Views besonders wichtig, da sie bestimmen, wie Daten abgefragt, wie Anfragen verarbeitet und wie Antworten generiert werden.

DRF APIVIEW

APIView in Django Rest Framework (DRF) ist eine klassenbasierte View, die für die Erstellung von Web-APIs konzipiert ist. Sie bietet die grundlegenden Bausteine, um eigene Views zu erstellen, die auf HTTP-Anfragen reagieren. APIView unterscheidet sich von Django's standardmäßigen View-Klassen durch die zusätzliche Funktionalität, die speziell für die Erstellung von APIs benötigt wird.

In APIView-basierten Klassen werden HTTP-Methoden wie GET, POST, PUT, DELETE usw. als Methoden der Klasse definiert. Dies macht es einfach, das Verhalten der View je nach Art der eingehenden Anfrage anzupassen.

Beispiel einer DRF APIView

```
from rest_framework.views import APIView
from rest_framework.response import Response
from .models import User
from .serializers import UserSerializer
from rest_framework import status


class UserListView(APIView):
    """
    API View, die eine Liste aller Benutzer zurückgibt.
    """
    def get(self, request):
        users = User.objects.all()
        serializer = UserSerializer(users, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
```

generische klassenbasierte Views

Diese generischen Views vereinfachen viele Routineaufgaben wie das Abrufen von Daten aus der Datenbank und das Serialisieren dieser Daten. Ein häufig verwendetes Beispiel ist die ListAPIView, die speziell für das Auflisten von Ressourcen konzipiert ist.

Jede generische View benötigt zumindest ein Queryset und eine Serializer-Klasse.

Beispiel generische ListAPIView

```
from rest_framework.generics import ListAPIView
from .models import User
from .serializers import UserSerializer


class UserListView(ListAPIView):
    """
    Generische API View, die eine Liste aller Benutzer zurückgibt.
    """
    queryset = User.objects.all()
    serializer_class = UserSerializer
```

Übersicht über wichtige generische Klassen

Klassenname	Beschreibung
ListAPIView	Für die Anzeige einer Liste von Objekten
RetrieveAPIView	Zum Abrufen eines einzelnen Objekts
CreateAPIView	Für das Erstellen neuer Objekte
DestroyAPIView	Zum Löschen eines Objekts
UpdateAPIView	Zum Aktualisieren bestehender Objekte
ListCreateAPIView	Eine Kombination aus List- und Create-API-View
RetrieveUpdateAPIView	Kombiniert das Abrufen und Aktualisieren in einer View
RetrieveDestroyAPIView	Eine Kombination aus Retrieve- und Destroy-View
RetrieveUpdateDestroyAPIView	Kombiniert Abrufen, Aktualisieren und Löschen

UrL-Konfiguration

In Django und im Django Rest Framework bezieht sich der Begriff "URLs" auf die URL-Konfigurationen, die definieren, wie URL-Pfade zu den entsprechenden Views in einer Django-Webanwendung oder einer API zugeordnet werden.

Jede URL-Konfiguration ordnet einen spezifischen URL-Pfad (oder Muster) einer View-Funktion oder einer View-Klasse zu, die dann die Anfragen bearbeitet, die an diesen Pfad gesendet werden. Diese Zuordnungen werden in der Regel in einer Datei namens urls.py innerhalb einer Django-App definiert.

Beispiel für URLs einer Event-API

URL	HTTP Methode	View	Beschreibung
/api/events/	GET	EventListView	Listet alle Events auf
/api/events/{id}	GET	EventDetailView	Listet einen Event auf
/api/events/	POST	EventCreateView	Legt neuen Event an
/api/events/{id}	DELETE	EventDeleteView	löscht Event
/api/events/{id}	PUT / PATCH	EventUpdateView	Editiert einen Event

Permissions

Permissions in Django Rest Framework (DRF) sind Regeln oder Bedingungen, die bestimmen, ob ein Benutzer auf eine bestimmte API-View zugreifen oder eine Aktion durchführen darf. Permissions sind essentiell für die Sicherheit und Kontrolle der Zugriffe in einer Webanwendung. DRF bietet eine Vielzahl vordefinierter Permissions, und Sie können auch eigene erstellen.

Permission-Klasse	Beschreibung
AllowAny	erlaubt jedem Benutzer Zugriff auf die View
IsAuthenticated	Erlaubt nur Zugriff für authentifizierte Benutzer
IsAdminUser	Benutzer mit is_staff bzw. is_superuser Status
IsAuthenticatedOrReadOnly	Authentifizierte Benutzer haben vollen Zugriff, während anonyme Benutzer nur Lesezugriff haben
DjangoModelPermissions	Verwendet die Standard-Modellberechtigungen von Django

Permissions nutzen

```
from rest_framework.permissions import IsAuthenticatedOrReadOnly


class MyView(APIView):
    permission_classes = [IsAuthenticatedOrReadOnly]
    # Ihre View-Logik
```

permission_classes ist eine Liste von Permissions für diese View. Befinden sich mehr als eine Permission in der Liste, gilt hier eine UND-Bedinung.