

rest framework

rest architecture

api = application programming interface -> we can run multiple app using one database. Example facebook whatsapp insta server down if the organisation use one database but they are interconnected each other through api. if server down then 3 application will not work properly

convert database in api

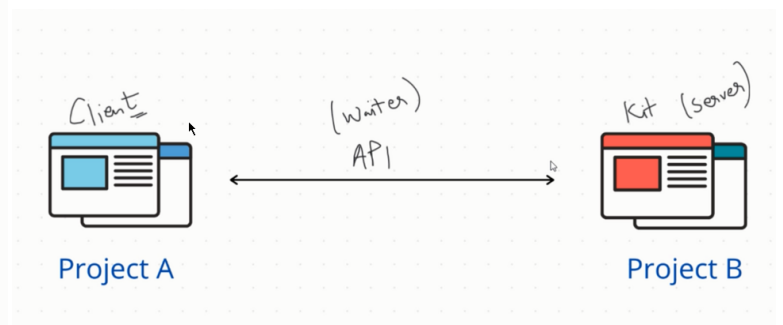
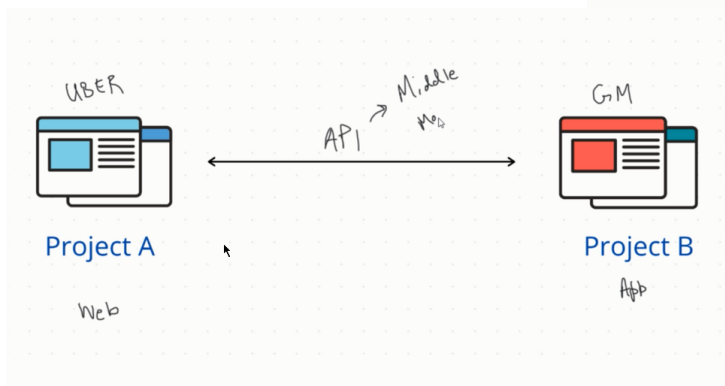
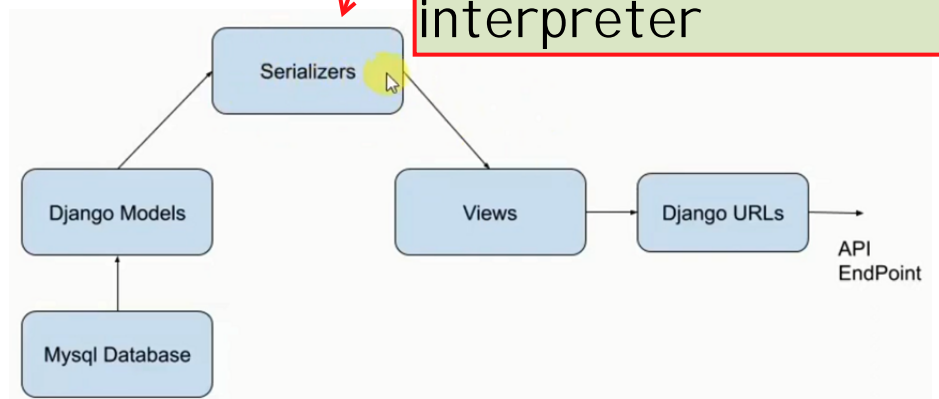
Rest = Representational state transfer

is a type of architecture (make of such kinds of algo) thats makes the data in one kind of standard data form JSON

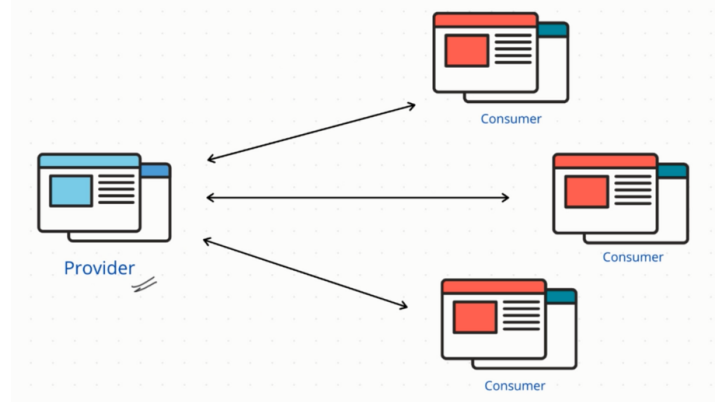
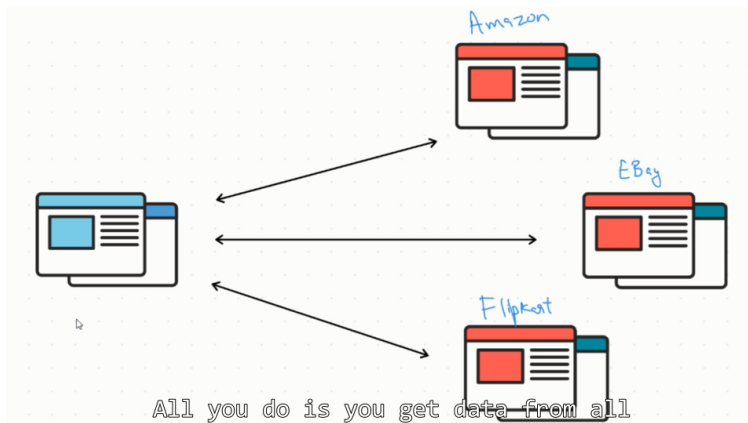
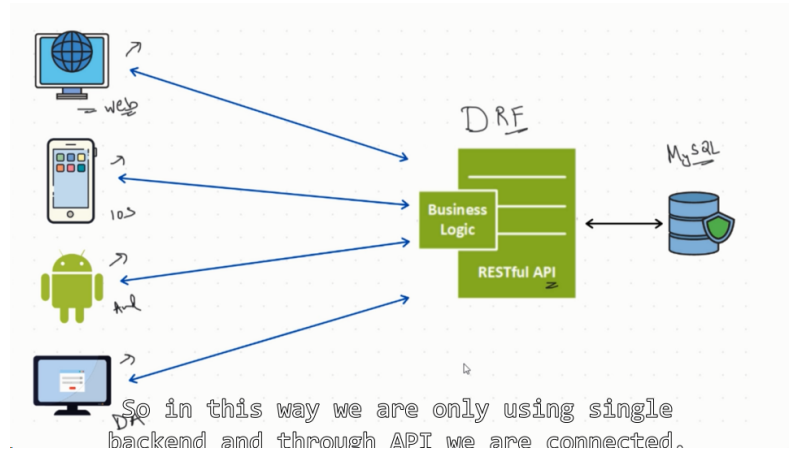
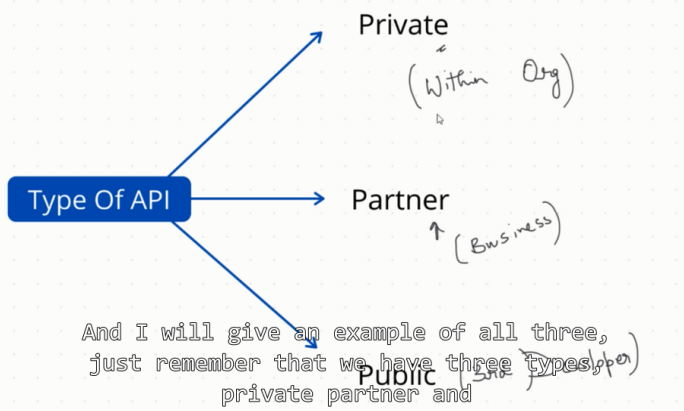
json=Java Script Object Notation

take data from model and convert in json.example chinese interpreter

user request-> view
->model->database
->view



api act as a middle man between client and server



Understanding URL

<https://www.api.movielist.com/movies/>
<https://www.api.movielist.com/movies/list/>

<https://www.api.movielist.com/movies/127/>
<https://www.api.movielist.com/movies/127/reviews/>
<https://www.api.movielist.com/movies/127/reviews/?limit=20>

<https://www.api.movielist.com/account/login/>
<https://www.api.movielist.com/account/register/>

It can be Spider-Man, Superman or any other movie.

Understanding URL

<https://www.api.movielist.com/movies/> *List*
<https://www.api.movielist.com/movies/list/>

<https://www.api.movielist.com/movies/127/> *Ind*
<https://www.api.movielist.com/movies/127/reviews/>
<https://www.api.movielist.com/movies/127/reviews/?limit=20>
Base URL
<https://www.api.movielist.com/account/login/>
<https://www.api.movielist.com/account/register/> *End Point*
 And then this remaining part, this part is known as End Point.

API REST → Rest API

(status code)

① endpoint
② method

③ headers
④ the data

json data

CRUD

- Create → POST
- Read → GET
- Update → PUT
- Delete → DELETE

HTTP Methods

So if I talk about them individually, we call them Create, retrieve, update and delete.

Understanding URL

→ <https://www.api.movielist.com/movies/>] LIST

GET

POST

→ <https://www.api.movielist.com/movies/127/>] IND

GET

PUT

DELETE

API + REST Architecture → REST API

1. End Points

2. Methods (CRUD)

3. Headers (Status Code)

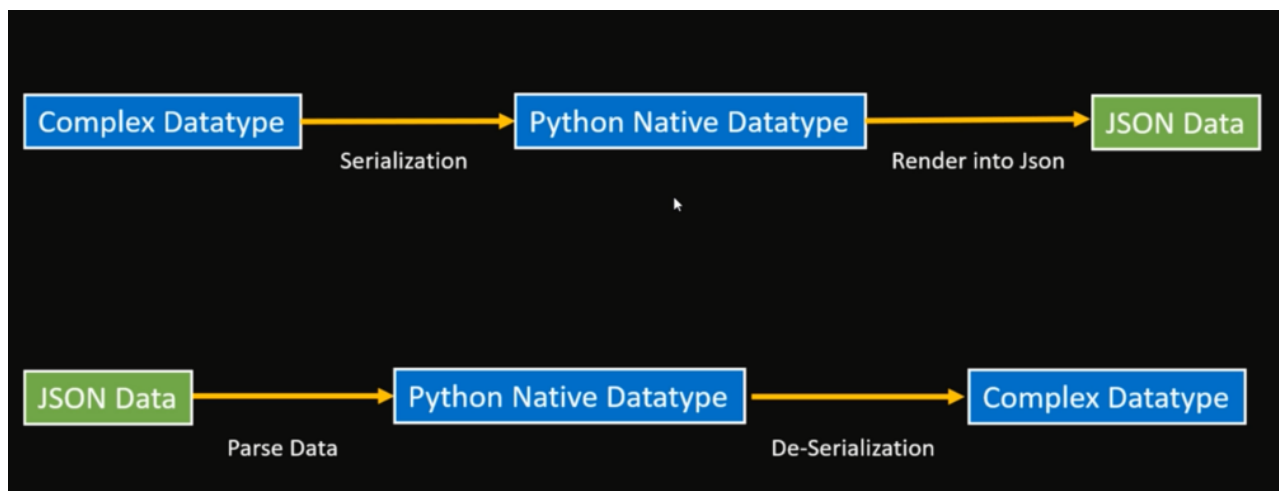
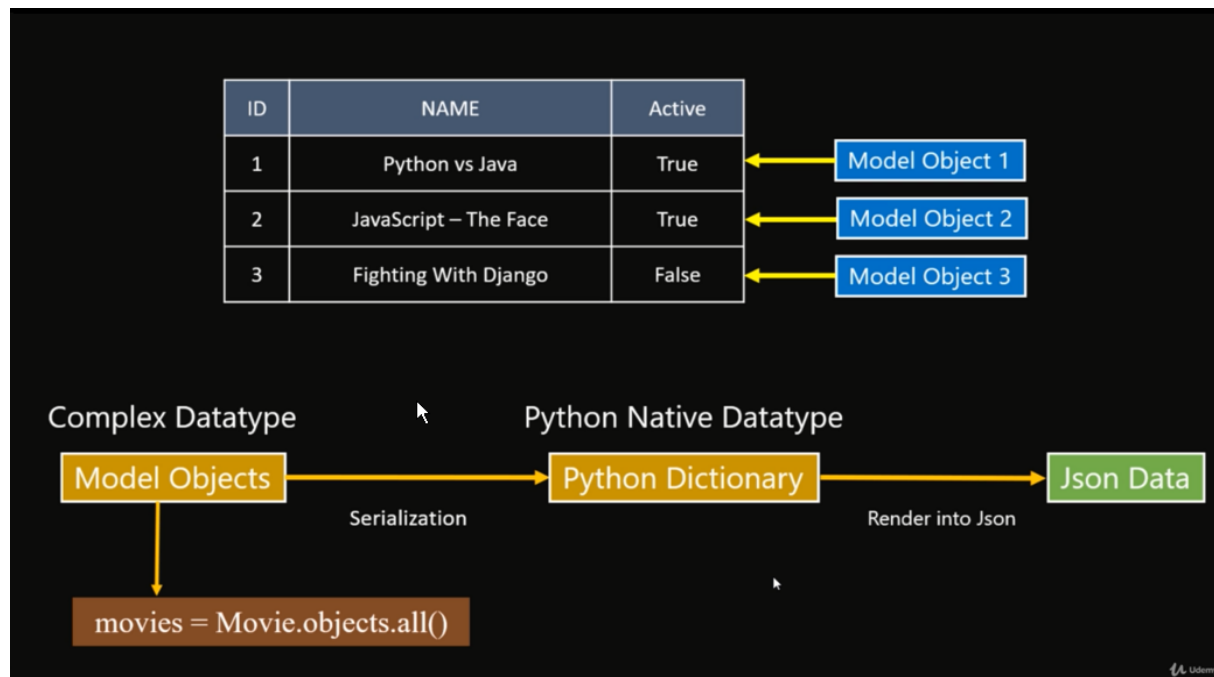
4. The Data (JSON)

```
def movie(request):
    movies = Movie.objects.all()
    data = {
        'movies': list(movies.values())
    }
    return JsonResponse(data)
```

queryset -> python
dictionary
python dictionary -> json
response

```
def movie_details(request, pk):
    movie = Movie.objects.get(id=pk)
    data = {
        'name' = movie.name,
        'description' = movie.description
    }
    return JsonResponse(data)
```

serializations in DRF

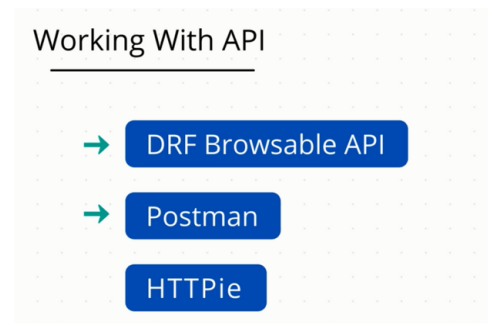
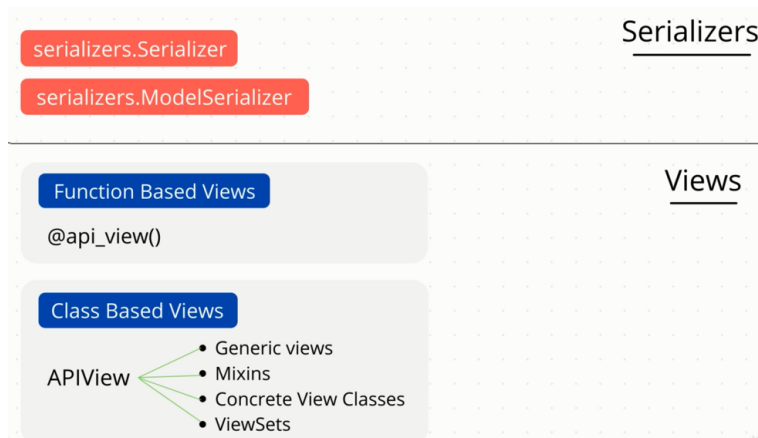


Type Of Serializers

Type Of Views

Working With API

serializers.Serializer
serializers.ModelSerializer



serializers:

```
class MovieSerializer(serializers.Serializer):
    id=serializers.IntegerField(read_only=True)
    name=serializers.CharField()
```

views:

```
@api_view(['GET','POST'])
def movie_list(request):
    movies=Movie.objects.all()
    serializer=MovieSerializer(movies, many=True)
    return Response(serializer.data)
```

```
@api_view(['GET','POST'])
def movie_details(request,pk):
    movie=Movie.objects.get(id=pk)
    serializer=MovieSerializer(movie)
    return Response(serializer.data)
```

views:

```
@api_view(['GET', 'POST'])
def movie_list(request):
    if request.method == 'GET':
        movies=Movie.objects.all()
        serializer=MovieSerializer(movies)
        return Response(serializer.data)

    if request.method == 'POST':
        serializer=MovieSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(serializer.errors)
```

```
@api_view(['GET', 'PUT', 'DELETE'])
def movie_details(request, pk):
    if request.method == "GET":
        movie=Movie.objects.get(id=pk)
        serializer=MovieSerializer(movie)
        return Response(serializer.data)

    if request.method=='PUT':
        movie=Movie.objects.get(id=pk)
        serializer=MovieSerializer(movie,data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            w return Response(serializer.errors)

    if request.method=="DELETE":
        movie=Movie.objects.get(id=pk)
        movie.delete()
        return Response(status=status.HTTP__204__No
Content)
```

status Code

Informational - 1xx

This class of status code indicates a provisional response. There are no 1xx status codes used in REST framework by default.

```
HTTP_100_CONTINUE  
HTTP_101_SWITCHING_PROTOCOLS
```

Successful - 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

```
HTTP_200_OK  
HTTP_201_CREATED  
HTTP_202_ACCEPTED  
HTTP_203_NON_AUTHORITATIVE_INFORMATION  
HTTP_204_NO_CONTENT  
HTTP_205_RESET_CONTENT  
HTTP_206_PARTIAL_CONTENT  
HTTP_207_MULTI_STATUS  
HTTP_208_ALREADY_REPORTED  
HTTP_226_IM_USED
```

Redirection - 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.

```
HTTP_300_MULTIPLE_CHOICES  
HTTP_301_MOVED_PERMANENTLY  
HTTP_302_FOUND  
HTTP_303_SEE_OTHER  
HTTP_304_NOT_MODIFIED  
HTTP_305_USE_PROXY  
HTTP_306_RESERVED  
HTTP_307_TEMPORARY_REDIRECT  
HTTP_308_PERMANENT_REDIRECT
```


Client Error - 4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

HTTP_400_BAD_REQUEST

HTTP_401_UNAUTHORIZED

HTTP_402_PAYMENT_REQUIRED

HTTP_403_FORBIDDEN

HTTP_404_NOT_FOUND

HTTP_405_METHOD_NOT_ALLOWED

HTTP_406_NOT_ACCEPTABLE

HTTP_407_PROXY_AUTHENTICATION_REQUIRED

HTTP_408_REQUEST_TIMEOUT

HTTP_409_CONFLICT

HTTP_410_GONE

HTTP_411_LENGTH_REQUIRED

HTTP_412_PRECONDITION_FAILED

HTTP_413_REQUEST_ENTITY_TOO_LARGE

HTTP_414_REQUEST_URI_TOO_LONG

HTTP_415_UNSUPPORTED_MEDIA_TYPE

HTTP_416_REQUESTED_RANGE_NOT_SATISFIABLE

HTTP_417_EXPECTATION_FAILED

HTTP_422_UNPROCESSABLE_ENTITY

HTTP_423_LOCKED

HTTP_424_FAILED_DEPENDENCY

HTTP_426_UPGRADE_REQUIRED

HTTP_428_PRECONDITION_REQUIRED

HTTP_429_TOO_MANY_REQUESTS

HTTP_431_REQUEST_HEADER_FIELDS_TOO_LARGE

HTTP_451_UNAVAILABLE_FOR_LEGAL_REASONS

Server Error - 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

HTTP_500_INTERNAL_SERVER_ERROR

HTTP_501_NOT_IMPLEMENTED

HTTP_502_BAD_GATEWAY

HTTP_503_SERVICE_UNAVAILABLE

HTTP_504_GATEWAY_TIMEOUT

HTTP_505_HTTP_VERSION_NOT_SUPPORTED

HTTP_506_VARIANT_ALSO_NEGOTIATES

HTTP_507_INSUFFICIENT_STORAGE

HTTP_508_LOOP_DETECTED

HTTP_509_BANDWIDTH_LIMIT_EXCEEDED

HTTP_510_NOT_EXTENDED

HTTP_511_NETWORK_AUTHENTICATION_REQUIRED

Helper functions

The following helper functions are available for identifying the category of the response code.

```
is_informational() # 1xx
is_success()      # 2xx
is_redirect()     # 3xx
is_client_error() # 4xx
is_server_error() # 5xx
```

```
@api_view(['GET','POST'])
def movie_list(request):
    if request.method=="get":
        movies=Movie.objects.all()
        serializer=MovieSerializer(movies,many=True)
        return Response(serializer.data)

    if request.method=="POST":
        serializer=MovieSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(status=status.HTTP__404)
```

```

@api_view(['GET', 'PUT', 'DELETE'])
def movie_details(request, pk):
    if request.method == 'GET':
        try:
            movie = Movie.objects.get(id=pk)
        except Movie.DoesNotExist:
            content = {'errors': 'Movie not found'}
            return Response(content, status=status.HTTP_404_NOT_FOUND)

        serializer = MovieSerializer(movie)
        return response(serializer.data)

    if request.method == "PUT":
        movie = Movie.objects.get(id=pk)
        serializer = MovieSerializer(movie, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

    if request.method == "DELETE":
        movie = Movie.objects.get(id=pk)
        movie.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

```

```
class MovieListAV(API View):
    def get(self, request):
        movies = Movie.objects.all()
        serializer = MovieSerializer(movies, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = MovieSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_201_CREATED)
        else:
            return Response(serializer.errors)
```

```
class MovieDetailsAV(API View):
    def get(self, request, pk):
        try:
            movie = Movie.objects.get(id=pk)
        except Movie.DoesNotExist:
            content = {
                'error': 'Movie not found'
            }
            return Response(content,
status=status.HTTP_400_BAD_REQUEST)
            serializer = MovieSerializer(movie)
            print(serializer)
            return Response(serializer.data)

    def put(self, request, pk):
        movie = Movie.objects.get(id=pk)
        serializer = MovieSerializer(movie, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
status=status.HTTP_202_ACCEPTED)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk):
        Movie.objects.get(id=pk).delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

validations:

field level

```
def validate_name(self, value):
    if len(value) < 2:
        raise serializers.ValidationError(
            'Name can not be less than 2')
    else:
        return value
```

object level

```
def validate(self, data):
    if data['name'] == data['description']:
        raise serializers.ValidationError(
            ('Name and Description can not same'))
    else:
        return data
```

validators

```
def length_name(value):
    if len(value) < 2:
        raise serializers.ValidationError('name can not
less than 2')
    else:
        return value

class MovieSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    name = serializers.CharField(validators=[length_name])
```

Serializer Fields and Core Arguments:

```
class MovieSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    name = serializers.CharField(validators=[length_name])
```

Model Serializer:

```
class MovieSerializer(serializers.ModelSerializer):
    class Meta:
        model = Movie
        fields = "__all__"
        # fields = ['id', 'name', 'description']
        # exclude = ['name']
```

```
def validate_name(self, value):
    if len(value) < 2:
        raise serializers.ValidationError('Name can not be less than 2')
    else:
        return value

def validate(self, data):
    if data['name'] == data['description']:
        raise serializers.ValidationError('Name and Description can not same')
    else:
        return data
```

custom Serializer:

```
class MovieSerializer(serializers.ModelSerializer):
    length_name=serializers.SerializerMethodField()
    class Meta:
        model=Movie
        fields="__all__"
    def get_length_name(self,object):
        return len(object.name)
```

Three kinds of connections:

one to one =

one to many =

many to many=

One To One Relations:

```
class Place(models.Model):
    location=models.CharField(max_length=70)

class Restaurant(models.Model):
    place=models.OneToOne(Place,on_delete=models.CASCADE)
```

One To Many:

```
class Reporter(models.Model):
    name=models.CharField(max_length=80)

class Article(models.Model):
    reporter=models.ForeignKey(Reporter)
```


Many To Many

```
class Publication(models.Model):  
    name=models.CharField(max_length=90)  
  
class Article(models.Model):  
    publication=models.ManyToManyField(Publication)
```

Nested Serializer->

```
class StreamingPlatformSerializer  
(serializers.ModelSerializer):  
    watchlist = WatchListSerializer(many=True,  
read_only=True)
```

```
class Meta:  
    model = StreamingPlatform  
    fields = "__all__"
```

```
class StreamPlatformSerializer(serializers.ModelSerializer):  
    # watchlist = WatchListSerializer(many=True, read_only=True)  
    watchlist = serializers.HyperlinkedRelatedField(  
        many=True,  
        read_only=True,  
        view_name='movie-detail'  
    )
```

```

class StreamPlatformDetailsAV(API View):
    def get(self, request, pk):
        try:
            stream_platform = StreamingPlatform.objects.get(id=pk)
        except StreamingPlatform.DoesNotExist:
            return Response(status=status.HTTP_204_NO_CONTENT)
        serializer = StreamingPlatformSerializer(stream_platform)
        return Response(serializer.data)

    def put(self, request, pk):
        stream_platform = StreamingPlatform.objects.get(id=pk)
        serializer = StreamingPlatformSerializer(stream_platform,data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk):
        StreamingPlatform.objects.get(id=pk).delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

```

```

class WatchListListAV(API View):
    def get(self, request):
        movies = WatchList.objects.all()
        serializer = WatchListSerializer(movies, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = WatchListSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        else:
            return Response(serializer.errors)

```

```

class WatchListDetailsAV(API View):
    def get(self, request, pk):
        try:
            movie = WatchList.objects.get(id=pk)
        except WatchList.DoesNotExist:
            content = {
                'error': 'not found'
            }
            return Response(content, status=status.HTTP_400_BAD_REQUEST)
        serializer = WatchListSerializer(movie)
        print(serializer)
        return Response(serializer.data)

    def put(self, request, pk):
        movie = WatchList.objects.get(id=pk)
        serializer = WatchListSerializer(movie, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_202_ACCEPTED)
        else:
            return Response(status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk):
        WatchList.objects.get(id=pk).delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

```

```
class ReviewList(mixins.ListModelMixin,
                 mixins.CreateModelMixin,
                 generics.GenericAPIView):
    queryset = Review.objects.all()
    serializer_class = ReviewSerializer

    def get(self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        return self.create(request, *args, **kwargs)
```

```
class ReviewDetail(mixins.RetrieveModelMixin,
                   mixins.UpdateModelMixin,
                   mixins.DestroyModelMixin,
                   generics.GenericAPIView):
    queryset = Review.objects.all()
    serializer_class = ReviewSerializer

    def get(self, request, *args, **kwargs):
        return self.retrieve(request, *args, **kwargs)

    def put(self, request, *args, **kwargs):
        return self.update(request, *args, **kwargs)

    def delete(self, request, *args, **kwargs):
        return self.destroy(request, *args, **kwargs)
```

Authentication manages user is logged in or not or valid user

Permission : user can access this url or not (any kinds of restriction)

