Challenge 2

Suggest improvements to the API design

To improve the API, I suggest the following, which I have given the sample code:

1. We can provide the nested serializer of “core” app as below:

serializer.py

```
from rest_framework import serializers
from rest_framework_dataclasses.serializers import DataclassSerializer

from .cdr import ChargeDetailRecord
from .rate import Rate, ChargingProcessResult


class ChargeDetailRecordSerializer(DataclassSerializer):
    class Meta:
        dataclass = ChargeDetailRecord


class RateSerializer(DataclassSerializer):
    class Meta:
        dataclass = Rate


class ChargingProcessSerializer(serializers.Serializer):
    cdr = ChargeDetailRecordSerializer(many=False)
    rate = RateSerializer(many=False)


class ComponentSerializer(DataclassSerializer):
    class Meta:
        dataclass = ChargingProcessResult


class ChargingProcessResultSerializer(serializers.Serializer):
    overall = serializers.FloatField()
    components = ComponentSerializer(many=False)
```

2. I put the main components calculations in utils folder as helper functions, named **rate_method.py** and **cdr_method.py**, we can move them to **rate.py** and **cdr.py** @dataclass definition files in ” core” app:

rate.py

```python
from dataclasses import dataclass

from .cdr import ChargeDetailRecord


@dataclass
class Rate:
    energy: float
    time: float
    transaction: float

    def calculate_cost(self, cdr: ChargeDetailRecord):
        energy = round(self.energy * cdr.total_energy, 3)
        time = round(cdr.charging_time * self.time, 3)
        transaction = round(self.transaction * 1, 3)
        return ChargingProcessResult(energy, time, transaction)


@dataclass
class ChargingProcessResult:
    energy: float
    time: float
    transaction: float

    @property
    def overall(self):
        return round(self.energy + self.time + self.transaction, 2)
```

cdr.py

```python
from dataclasses import dataclass
from datetime import datetime


@dataclass
class ChargeDetailRecord:
    meterStart: int
    timestampStart: datetime
    meterStop: int
    timestampStop: datetime

    @property
    def total_energy(self):
        return (self.meterStop - self.meterStart)/1000

    @property
    def charging_time(self):
        return (self.timestampStop - self.timestampStart).seconds / 3600
```

3. With these changes, our **views.py** in “core” app changes to the following and returns the result:

```python
from rest_framework import status
from rest_framework.response import Response
from rest_framework.views import APIView

from .serializers import ChargingProcessSerializer, ChargingProcessResultSerializer


class ChargingProcessRating(APIView):

    def post(self, request):
        serializer = ChargingProcessSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        cdr = serializer.validated_data["cdr"]
        rate = serializer.validated_data["rate"]

        result = rate.calculate_cost(cdr)
        output_data = ChargingProcessResultSerializer(
            {
                'overall': result.overall,
                'components': result
            }
        )

        return Response(output_data.data, status=status.HTTP_200_OK)
```