**Top up Serializer**

from rest\_framework import serializers

from ..models.transactionModel import Transaction

class TopupSerializer(serializers.ModelSerializer):

user\_id = serializers.IntegerField(*write\_only*=True)

project\_id = serializers.IntegerField(*write\_only*=True)

class Meta:

model = Transaction

fields = ['id', 'user\_id', 'project\_id', 'amount', 'payment\_method', 'transaction\_type', 'status', 'transaction\_date']

read\_only\_fields = ['id', 'transaction\_type', 'status', 'transaction\_date']

**Project Detail Serializer**

from rest\_framework import serializers

from ..models.farmerModel import Farmer

from ..models.financialReportModel import FinancialReport

from ..models.projectModel import Project

from django.db.models import Avg

class FarmerSerializer(serializers.ModelSerializer):

class Meta:

model = Farmer

fields = ['id', 'name', 'photoProfile']

*# Dalam FinancialReportSerializer*

class FinancialReportSerializer(serializers.ModelSerializer):

file\_url = serializers.SerializerMethodField()

class Meta:

model = FinancialReport

fields = ['id', 'file\_name', 'profit', 'file\_url']

def get\_file\_url(*self*, *obj*):

return *obj*.file.url *# Mengambil URL file dari ImageField/FileField*

class ProjectDetailSerializer(serializers.ModelSerializer):

status\_display = serializers.SerializerMethodField()

type\_display = serializers.SerializerMethodField()

profit = serializers.SerializerMethodField()

class Meta:

model = Project

fields = ['id', 'name', 'logo', 'type\_display', 'location', 'target\_funds', 'invested\_amount', 'status\_display', 'description', 'start\_date', 'end\_date', 'projectImage', 'profit']

def get\_status\_display(*self*, *obj*):

return *obj*.get\_status\_display()

def get\_type\_display(*self*, *obj*):

return *obj*.get\_type\_display()

def get\_profit(*self*, *obj*):

*# Menghitung rata-rata profit dari laporan keuangan terkait*

average\_profit = *obj*.financial\_reports.aggregate(Avg('profit'))['profit\_\_avg']

return round(average\_profit, 2) if average\_profit is not None else 0.0

**Project Serializer**

from rest\_framework import serializers

from ..models.projectModel import Project

from django.db.models import Avg

class ProjectSerializer(serializers.ModelSerializer):

status\_display = serializers.SerializerMethodField()

type\_display = serializers.SerializerMethodField()

profit = serializers.SerializerMethodField()

class Meta:

model = Project

fields = ['id', 'logo', 'name', 'type\_display', 'location','status\_display', 'profit', 'invested\_amount']

def get\_status\_display(*self*, *obj*):

return *obj*.get\_status\_display()

def get\_type\_display(*self*, *obj*):

return *obj*.get\_type\_display()

def get\_profit(*self*, *obj*):

*# Menghitung rata-rata profit dari laporan keuangan terkait*

average\_profit = *obj*.financial\_reports.aggregate(Avg('profit'))['profit\_\_avg']

return round(average\_profit, 2) if average\_profit is not None else 0.0

**Project Model**

from django.db import models

from django.core.validators import MinValueValidator, MaxValueValidator

from .farmerModel import Farmer

from django.db.models import Sum

class Project(models.Model):

STATUS\_CHOICES = [

('ongoing', 'Sedang berlangsung'),

('available', 'Tersedia'),

('not\_available', 'Tidak tersedia'),

]

TYPE\_PROJECT = [

('agriculture', 'Pertanian'),

('fishery', 'Perikanan'),

('farm', 'Peternakan'),

]

logo = models.ImageField(*upload\_to*='projectLogos/', *blank*=True, *null*=True)

name = models.CharField(*max\_length*=255)

type = models.CharField(*max\_length*=20, *choices*=TYPE\_PROJECT, *default*='agriculture')

projectImage = models.ImageField(*upload\_to*='projectImages/', *blank*=True, *null*=True)

location = models.CharField(*max\_length*=255)

invested\_amount = models.DecimalField(

*max\_digits*=15,

*decimal\_places*=2,

*default*=0.00

)

target\_funds = models.DecimalField(*max\_digits*=15, *decimal\_places*=2, *validators*=[MinValueValidator(0)])

status = models.CharField(*max\_length*=20, *choices*=STATUS\_CHOICES, *default*='not\_available')

description = models.TextField()

start\_date = models.DateField(*blank*=True, *null*=True)

end\_date = models.DateField(*blank*=True, *null*=True)

farmer = models.ForeignKey(Farmer, *on\_delete*=models.CASCADE, *related\_name*='projects')

def update\_invested\_amount(*self*):

"""

Metode untuk memperbarui total dana terkumpul berdasarkan tabel Portfolio.

"""

total\_invested = *self*.portfolios.aggregate(Sum('invested\_amount'))['invested\_amount\_\_sum'] or 0

*self*.invested\_amount = total\_invested

*self*.save()

def \_\_str\_\_(*self*):

return *self*.name

**Transaction Model**

from django.db import models

from django.utils.timezone import now

from django.contrib.auth.models import User

from .projectModel import Project

from .portfolioModel import Portfolio

class Transaction(models.Model):

PAYMENT\_METHODS = [

('bank\_transfer', 'Bank Transfer'),

('e\_wallet', 'E-Wallet'),

('credit\_card', 'Credit Card'),

]

STATUS\_CHOICES = [

('pending', 'Menunggu'),

('success', 'Berhasil'),

('failed', 'Gagal'),

]

TRANSACTION\_TYPE = [

('withdraw', 'Tarik Dana'),

('deposit', 'Setor Dana'),

]

user = models.ForeignKey(User, *on\_delete*=models.CASCADE, *related\_name*='transactions')

portfolio = models.ForeignKey(Portfolio, *on\_delete*=models.CASCADE, *related\_name*='transactions', *null*=True, *blank*=True)

project = models.ForeignKey(Project, *on\_delete*=models.CASCADE, *related\_name*='transactions')

amount = models.DecimalField(*max\_digits*=15, *decimal\_places*=2)

payment\_method = models.CharField(*max\_length*=20, *choices*=PAYMENT\_METHODS)

transaction\_type = models.CharField(*max\_length*=10, *choices*=TRANSACTION\_TYPE)

status = models.CharField(*max\_length*=20, *choices*=STATUS\_CHOICES, *default*='success')

transaction\_date = models.DateTimeField(*default*=now)

updated\_at = models.DateTimeField(*auto\_now*=True)

def \_\_str\_\_(*self*):

return f"Transaction {*self*.id} - {*self*.user.username} - {*self*.amount} ({*self*.transaction\_type})"

class Meta:

ordering = ['-transaction\_date']

**Transaction View**

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from rest\_framework.decorators import api\_view

from django.db.models import Sum

from django.utils.timezone import now

from ..serializers.topupSerializer import TopupSerializer

from django.contrib.auth.models import User

from ..models.projectModel import Project

from ..models.transactionModel import Transaction

from ..models.portfolioModel import Portfolio

from ..serializers.getTransactionSerializer import GetTransactionSerializer

class TopUpView(APIView):

def post(*self*, *request*):

serializer = TopupSerializer(*data*=*request*.data)

if serializer.is\_valid():

user\_id = *request*.data.get('user\_id')

project\_id = *request*.data.get('project\_id')

amount = *request*.data.get('amount')

payment\_method = *request*.data.get('payment\_method')

try:

user = User.objects.get(*id*=user\_id)

project = Project.objects.get(*id*=project\_id)

except (User.DoesNotExist, Project.DoesNotExist):

return Response({"error": "User or Project not found."}, *status*=status.HTTP\_404\_NOT\_FOUND)

*# Update total dana terkumpul di Project*

total\_funds = Portfolio.objects.filter(*project*=project).aggregate(Sum('invested\_amount'))['invested\_amount\_\_sum'] or 0

total\_funds += amount

project.update\_invested\_amount()

*# Buat atau update Portfolio*

try:

portfolio = Portfolio.objects.get(*user*=user, *project*=project)

portfolio.invested\_amount += amount

portfolio.last\_updated = now()

portfolio.ownership\_percentage = ((portfolio.invested\_amount + amount) / total\_funds) \* 100

portfolio.save()

except Portfolio.DoesNotExist:

Portfolio.objects.create(

*user*=user,

*project*=project,

*invested\_amount*=amount,

*ownership\_percentage*= (amount / total\_funds) \* 100

)

*# Update ownership\_percentage untuk semua portfolio di proyek ini*

for p in Portfolio.objects.filter(*project*=project):

p.ownership\_percentage = (p.invested\_amount / total\_funds) \* 100

p.save()

*# Buat transaksi baru*

transaction = Transaction.objects.create(

*user*=user,

*project*=project,

*portfolio*=portfolio,

*amount*=amount,

*payment\_method*=payment\_method,

*transaction\_type*='deposit',

*status*='success',

)

return Response({

"message": "Top-up transaction and portfolio updated successfully.",

"transaction": TopupSerializer(transaction).data

}, *status*=status.HTTP\_201\_CREATED)

else:

return Response(serializer.errors, *status*=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET'])

def userTransactions(*request*, *userId*):

try:

*# Ambil transaksi user dengan optimalisasi query*

transactions = Transaction.objects.filter(*user\_id*=*userId*).select\_related('project').order\_by('-transaction\_date')

*# Jika tidak ada transaksi, kembalikan pesan*

if not transactions.exists():

return Response({"message": "No transactions found for this user."}, *status*=status.HTTP\_404\_NOT\_FOUND)

*# Serialize data transaksi*

serializer = GetTransactionSerializer(transactions, *many*=True)

return Response(serializer.data, *status*=status.HTTP\_200\_OK)

except Exception as e:

*# Tangani error dengan pesan error generik*

return Response({"error": str(e)}, *status*=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)

**Portfolio Model**

from django.db import models

from ..models.userModel import User

from ..models.projectModel import Project

class Portfolio(models.Model):

user = models.ForeignKey(

User,

*on\_delete*=models.CASCADE,

*related\_name*='portfolios'

)

project = models.ForeignKey(

Project,

*on\_delete*=models.CASCADE,

*related\_name*='portfolios'

)

ownership\_percentage = models.DecimalField(

*max\_digits*=5,

*decimal\_places*=2,

*null*=True,

*blank*=True

)

invested\_amount = models.DecimalField(

*max\_digits*=15,

*decimal\_places*=2

)

created\_at = models.DateTimeField(*auto\_now\_add*=True)

last\_updated = models.DateTimeField(*auto\_now*=True)

def \_\_str\_\_(*self*):

return f"Portfolio {*self*.user.username} - {*self*.project.name} - {*self*.invested\_amount}"

class Meta:

constraints = [

models.UniqueConstraint(

*fields*=['user', 'project'],

*name*='unique\_user\_project\_transaction'

)

]