RESTful web service

A Workshop on Django web application development

https://github.com/WasinTh/tutorial-django-rest

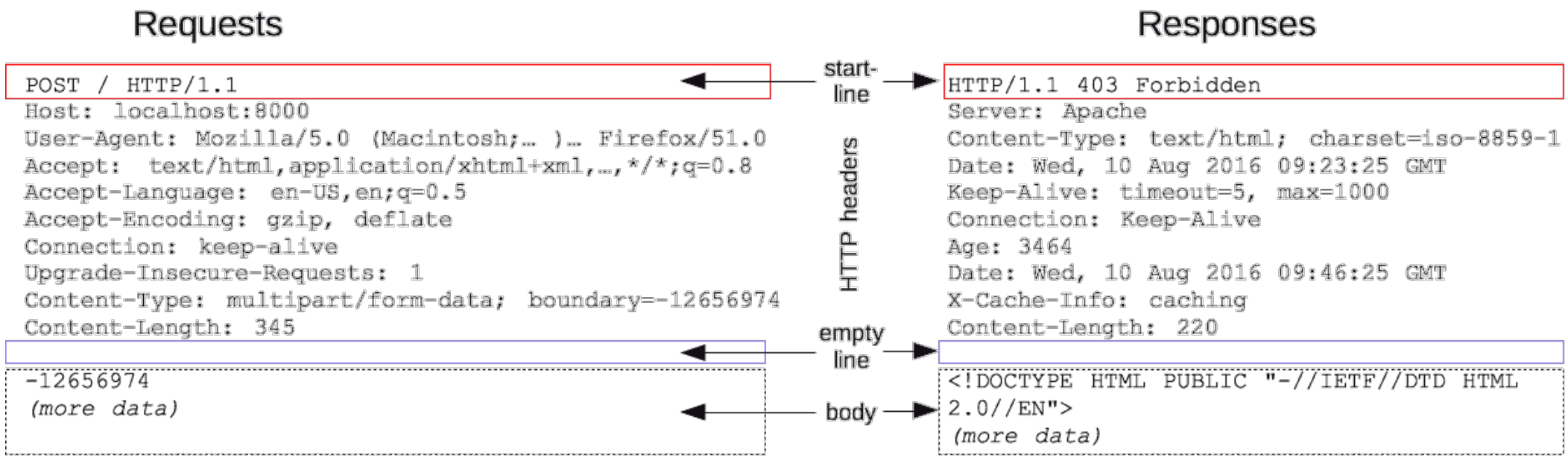
What is RESTful web service?

- Representational State Transfer (REST) คือรูปแบบพัฒนาโปรแกรม Web Application
  - ทำงานอยู่บน HTTP Protocol
  - ใช้สำหรับแยกจัดการ ส่วนประมวลผลข้อมูล (Server) และส่วนแสดงผล (Client) ออกจากกัน
  - โปรแกรมทั้งฝั่ง Client และ Server ต้องสามารถเข้าใจข้อตกลงที่จะรับ-ส่งถึงกัน
  - โปรแกรมทั้งสองฝั่งสามารถพัฒนาด้วยภาษาอะไรก็ได้ เช่น

- Back-End
  - Django
  - Node.js
  - PHP
  - .NET

- Front-End
  - React
  - Vue
  - Angular
  - Flutter

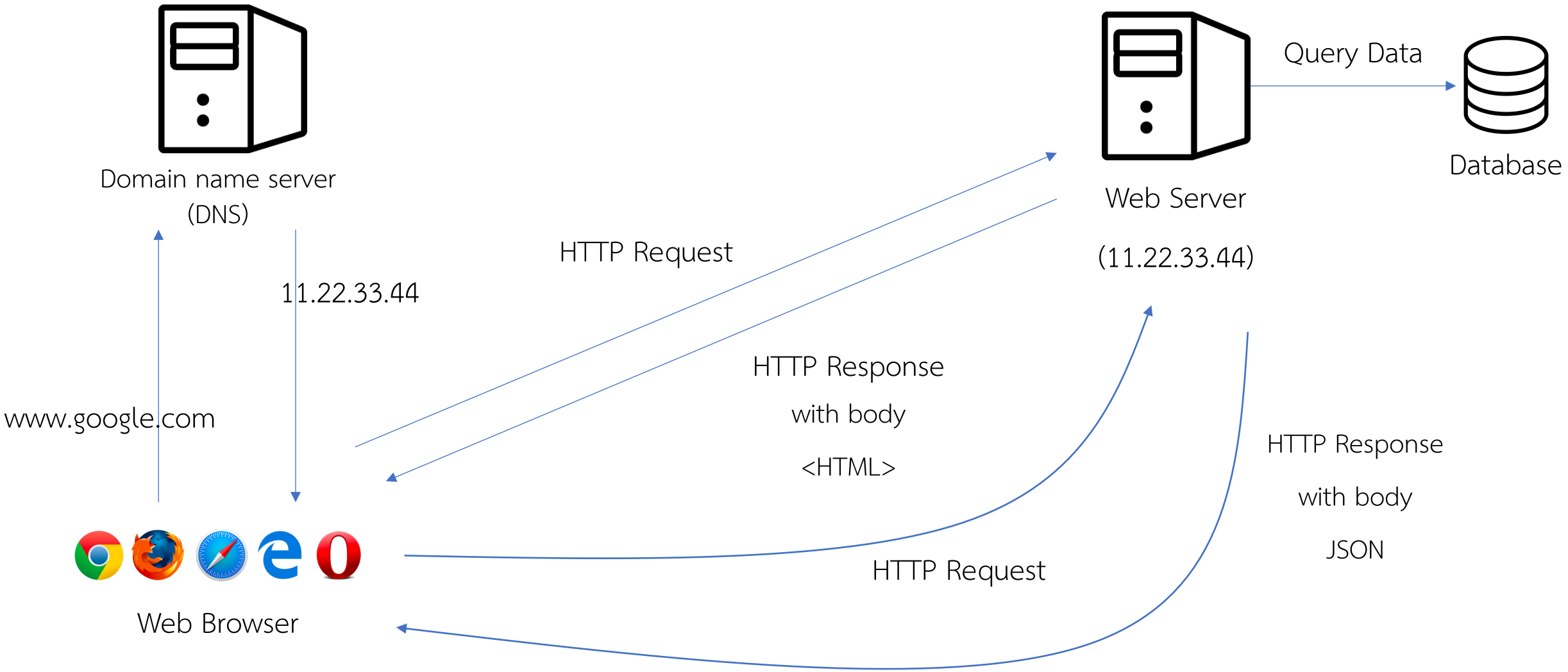
HTTP Message



ข้อตกลงของ RESTful web service

- HTTP headers ประกอบไปด้วย GET, POST, PUT, PATCH and DELETE
  - GET : ดึงข้อมูลจาก Server
  - POST : สร้างข้อมูลบน Server
  - PUT/PATCH : แก้ไขข้อมูลบน Server
  - DELETE : ลบข้อมูลบน Server
- HTTP Body ทั้งส่วนของ request และ response จะอยู่ในรูปแบบของ JSON

RESTful Web Technology



Django RESTful web service – T003

account/views.py

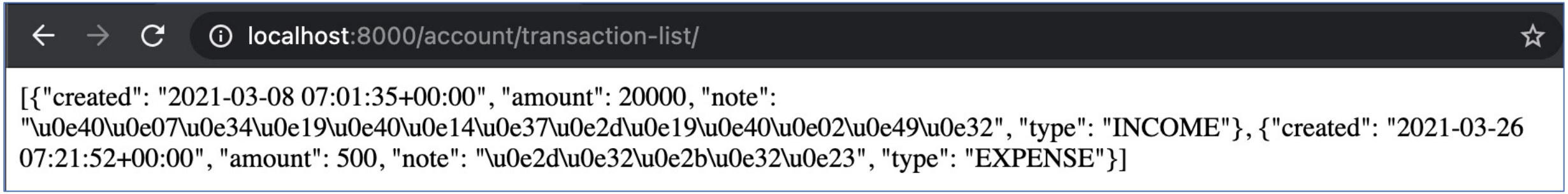
```
Import json
from django.http import HttpResponse
from account.models import Transaction


def current_balance_view(request):
    data = {'balance': 0}
    for transaction in Transaction.objects.all():
        if transaction.type == Transaction.Type.INCOME:
            data['balance'] += transaction.amount
        else:
            data['balance'] -= transaction.amount

    return HttpResponse(json.dumps(data))
```

List Transaction REST API – T004

- สร้าง URL ใหม่สำหรับเรียกแสดง List ของ Transaction ทั้งหมด
- Example URL
  - http://localhost:8000/account/transaction-list/



Django REST framework

- Framework สำหรับสร้าง REST API โดยเฉพาะ
- ข้อดีของ Django REST framework เช่น
  - ตรวจสอบความถูกต้องของ parameter ที่ส่งเข้ามาใน server โดยอัตโนมัติ
  - มีหน้า web สำหรับทดสอบ API
  - มีกระบวนการ Serialization สำหรับการแปลงค่าระหว่าง Model Object และ JSON
  - สามารถสร้าง Document ได้โดยอัตโนมัติ
- ข้อเสีย
  - เป็น Framework ที่มีขนาดใหญ่ และมี plugin ย่อย อีกเป็นจำนวนมาก

Installation

```
$ pip install djangorestframework
```

- Add 'rest_framework' to INSTALLED_APPS in settings.py

personal_finance/settings.py

```
INSTALLED_APPS = (
    ...
    “rest_framework”,
)
```

ทำความรู้จักกับ Serializer – T005

- Serializer เป็นตัวกลางสำหรับแปลงค่า JSON ให้เป็น Django Object ไปมา

account/serializers.py

```
import datetime
from rest_framework import serializers
from account.models import Transaction


class TransactionSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    created = serializers.DateTimeField(default=datetime.datetime.now())
    amount = serializers.IntegerField()
    note = serializers.CharField(required=False, allow_blank=True, default='')
    type = serializers.ChoiceField(choices=Transaction.Type)

    def create(self, validated_data):
        return Transaction.objects.create(**validated_data)
```

ทำความรู้จัก Serializer (2)

- รัน python manage.py shell แล้วทดสอบคำสั่งต่อไปนี้

```
from account.models import Transaction
from account.serializers import TransactionSerializer
t = Transaction.objects.last()
serializer = TransactionSerializer(t)
print(serializer.data)
```

- เราสามารถ serialize queryset ได้ เช่นกัน

```
serializer = TransactionSerializer(Transaction.objects.all(), many=True)
print(serializer.data)
```

ทำความรู้จัก Serializer (3)

- Serializer สามารถใช้ในการสร้าง object จาก JSON ได้

```
data = {'amount': 1000, 'type': Transaction.Type.INCOME}
Transaction.objects.count()
serializer = TransactionSerializer(data=data)
Transaction.objects.count()
serializer.is_valid()
serializer.save()
Transaction.objects.count()
```

ทดลองใช้ ModelSerializer

- จากตัวอย่าง TransactionSerializer ก่อนหน้านี้ จะเห็นว่า code ที่เขียนมีความคล้ายคลึงกับ Model เป็นอย่างมาก
- DjangoRestFramework ได้ออกแบบ ModelSerializer เพื่อลดการเขียน code เหล่านี้ลง

account/serializers.py

```
import datetime
from rest_framework import serializers
from account.models import Transaction


class TransactionSerializer(serializers.ModelSerializer):
    created = serializers.DateTimeField(default=datetime.datetime.now())
    type_display = serializers.CharField(source='get_type_display', read_only=True)

    class Meta:
        model = Transaction
        fields = ['id', 'created', 'amount', 'note', 'type', 'type_display']
```

นำ Serializer มาใช้กับ Views – T006

account/views.py

```
from rest_framework.response import Response
from rest_framework.decorators import api_view
from account.models import Transaction
from account.serializer import TransactionSerializer


@api_view(['GET'])
def transaction_list_view(request):
    serializer = TransactionSerializer(Transaction.objects.all(), many=True)
    return Response(data=serializer.data)
```

การใช้งาน Django REST framework

- Function Based Views จากตัวอย่างก่อนหน้า ใช้งานผ่านทาง decorator @api_view
- Class Based Views
  - APIView
  - generics API views
  - Viewsets

Class Based Views

- เป็นการ Upgrade Function-based view เพื่อให้รองรับการทำงานที่มากขึ้น
  - การจัดการ Authentication และ permission
  - มีการจัดการ Return response ที่เหมาะสมให้อัตโนมัติ
- มีการแทนที่ Django HTTPRequest และ HTTPResponse ด้วย rest_framework เช่นกัน

Class based view – T007

account/views.py

```
from rest_framework.views import APIView

class TransactionView(APIView):
    def get(self, request):
        serializer = TransactionSerializer(Transaction.objects.all(), many=True)
        return Response(data=serializer.data)
```

personal_finance/urls.py

```
from account.views import TransactionView
urlpatterns = [
    .........
    path('account/transaction/', TransactionView.as_view()),
]
```

การใช้งาน Django REST framework

- Function Based Views จากตัวอย่างก่อนหน้า ใช้งานผ่านทาง decorator @api_view
- Class Based Views
  - APIView
  - generics API views
  - Viewsets

Generics API View

- ปรับปรุงการใช้งานแบบเดิม โดยผ่าน Serializer
- ช่วยลดงานการเขียนโปรแกรมลง โดยการทำงานบางอย่างให้อัตโนมัติ เช่น
  - สร้าง JSON จาก Model โดยอัตโนมัติ
  - ตรวจสอบความถูกต้องของ JSON message ที่รับมาจาก client โดยอัตโนมัติ
- Generics API View ประกอบไปด้วย
  - ListAPIView รับ GET เพื่อแสดง list ของ object (GET) เพียงอย่างเดียว
  - CreateAPIView รับ POST สำหรับการสร้าง object
  - UpdateAPIView รับ PUT หรือ PATCH สำหรับ update ค่าของ object
  - DestroyAPIView รับ DELETE สำหรับ ลบ object

Generics API View (2)

- Class ที่สืบทอดจาก Generics API View จำเป็นต้องประกาศ property 2 ตัว คือ
  - queryset สำหรับใช้ในการ query object ที่ต้องการส่งไปให้ client
  - serializer_class สำหรับประกาศวิธีการแปลงค่าระหว่าง object และ JSON

account/views.py

```
from rest_framework import generics

class TransactionView(generics.ListAPIView):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer
```

Generics API View (3) – T008

- เราสามารถ Mix & Match generics views หลายๆ ตัวเข้าด้วยกันได้ เช่น URL สำหรับ List และ Create ตามตัวอย่างต่อไปนี้ จากนั้นดูความแตกต่างที่หน้า website

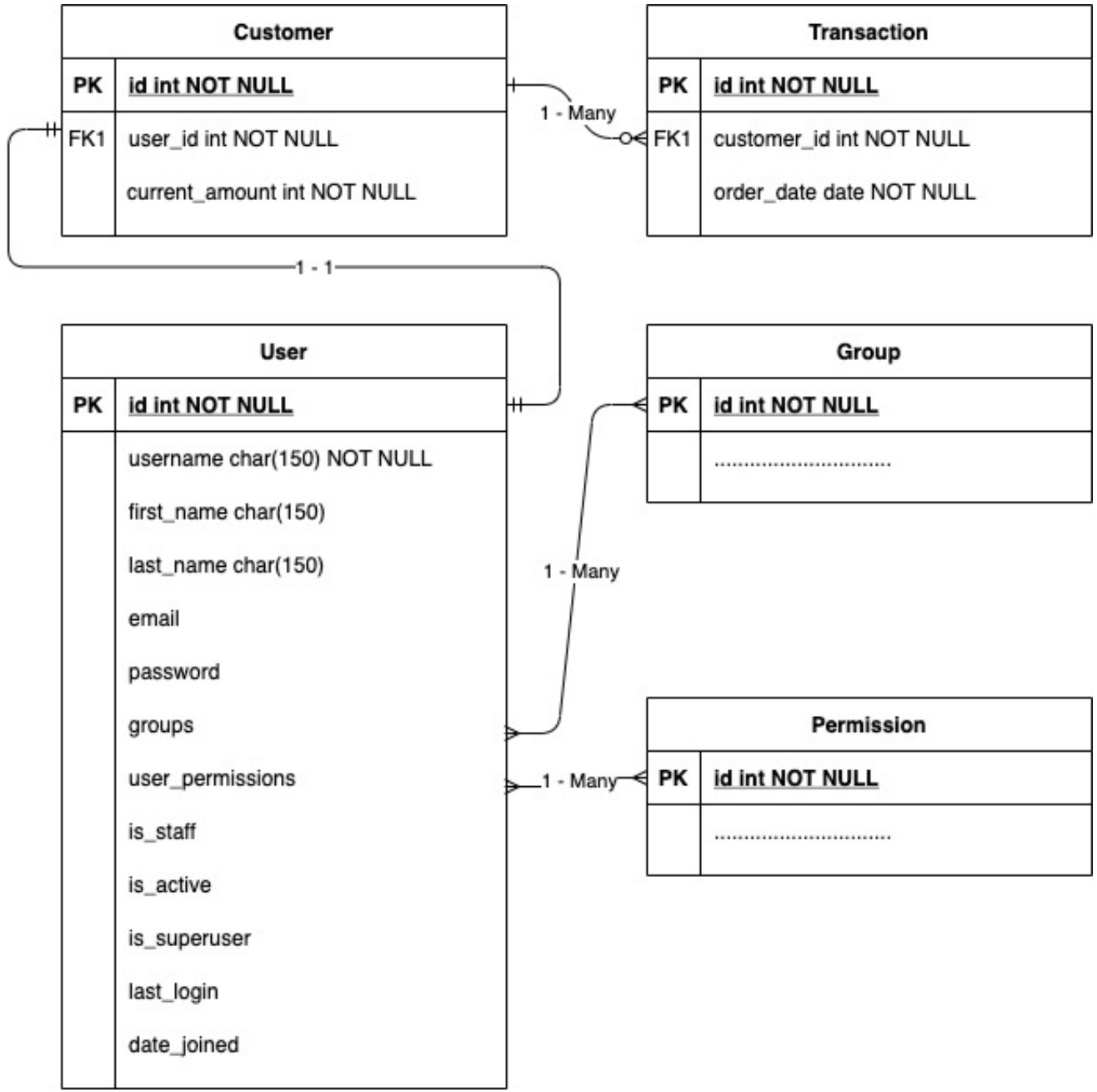
account/views.py

```
from rest_framework import generics

class TransactionView(generics.ListCreateAPIView):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer
```

Our API Problems

- ปัญหาของ Code ในปัจจุบัน ได้แก่
  - การฝากและการถอนเป็นของ user เพียงแค่คนเดียว
  - การคำนวณจำนวนยอดเงินปัจจุบันต้องอาศัยการ Query จากฐานข้อมูลทุก row
- ควรสร้าง Class ใหม่สำหรับเก็บข้อมูลของผู้ใช้งานปัจจุบันและเก็บข้อมูลของยอดเงินปัจจุบัน


Customer
PK
id int NOT NULL
FK1
user_id int NOT NULL
current_amount int NOT NULL
1 - Many
Transaction
PK
id int NOT NULL
FK1
customer_id int NOT NULL
order_date date NOT NULL
1 - 1
User
PK
id int NOT NULL
username char(150) NOT NULL
first_name char(150)
last_name char(150)
email
password
groups
user_permissions
is_staff
is_active
is_superuser
last_login
date_joined
Group
PK
id int NOT NULL
.....
1 - Many
Permission
PK
id int NOT NULL
.....
1 - Many

แก้ไข Models.py ให้ตรงกับที่ออกแบบไว้

account/models.py

```
from django.db import models
from django.contrib.auth import get_user_model

User = get_user_model()


class Customer(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    current_amount = models.IntegerField(default=0)


class Transaction(models.Model):

    class Type(models.TextChoices):
        INCOME = 'INCOME', 'รายรับ'
        EXPENSE = 'EXPENSE', 'รายจ่าย'

 created = models.DateTimeField()
    amount = models.IntegerField()
    note = models.TextField()
    type = models.CharField(max_length=32, choices=Type.choices)
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
```

Migrate Database – T009

- ลบไฟล์ db.sqlite3
- ลบไฟล์ account/migrations/0001_initial.py
- รันคำสั่งต่อไปนี้

```
python manage.py makemigrations
python manage.py migrate
python manage.py createsuperuser
```

- แก้ไข admin.py ให้สามารถเพิ่ม Customer และ Transaction ได้

สร้าง REST API สำหรับจัดการ Customer – T010

- เมื่อเรียก URL /account/customer จะต้องสามารถเห็น list ของ customer และสร้าง customer ใหม่ได้
- แก้ไขไฟล์ serializers.py, views.py, และ urls.py

TIPS: serializers.py

```
class xxxxxSerializer(serializers.ModelSerializer):
    class Meta:
        model = xxxxx
        fields = '__all__'
```

สรุป Customer Serializer

```
class CustomerSerializer(serializers.ModelSerializer):
    name = serializers.CharField(source='user.username', read_only=True)

    class Meta:
        model = Customer
        fields = '__all__'
```

การใช้งาน Django REST framework

- Function Based Views
- Class Based Views
  - APIView
  - generics API views
  - Viewsets

ViewSets & Routers

- ViewSets เป็น class สำหรับ views.py โดย Django REST Framework มีการแนบ operation ต่าง ๆ (GET, POST, PUT, PATCH, DELETE) ให้โดยอัตโนมัติ
- Routers เป็นส่วนที่ทำงานร่วมกับ ViewSets ภายในไฟล์ urls.py เพื่อลดความซับซ้อนของไฟล์

ViewSets & Router – ทดลองสร้าง ViewSets – T011

account/views.py

```
from rest_framework import viewsets

class TransactionViewSet(viewsets.ModelViewSet):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer


class CustomerViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer
```

account/urls.py

```
from django.urls import include
from rest_framework.routers import DefaultRouter
from account.views import TransactionViewSet, CustomerViewSet

router = DefaultRouter()
router.register('account/transaction-viewsets', TransactionViewSet)
router.register('account/customer-viewsets', CustomerViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

สรุปการใช้งาน Django REST framework

- Function Based Views - ไม่ควรใช้
- Class Based Views
  - APIView – ใช้กรณีที่ API ไม่เกี่ยวข้องหรือมีความใกล้เคียงกับ Models ที่ออกแบบไว้น้อย
  - generics API views – ใช้กรณี API มีความใกล้เคียงกับ Models แต่ต้องการใช้แค่บาง method
  - Viewsets – ใช้กรณี API มีความใกล้เคียงกับ Models และต้องการใช้ method หลาย ๆ ตัว

Authorization & Authentication

Authorization & Authentication

- ระบบปัจจุบันอนุญาตให้ใครก็ได้สามารถเข้ามายิง API ได้ เราสามารถเพิ่ม Permission ในการเข้าถึง API ได้ ดังต่อไปนี้
- เพิ่ม code ใน views.py แล้วทดสอบ /account/transaction ใน Chrome Incognito Mode

account/views.py

```
from rest_framework import generics
from rest_framework import permissions


class TransactionView(generics.ListAPIView):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
```

Authorization & Authentication (2) – T012

- กรณีต้องการตั้งค่า Permission เหมือนกันทั้งระบบ สามารถตั้งค่าได้ผ่านทาง settings.py
- ทดสอบเพิ่ม code นี้ลงไปใน settings.py

settings.py

```
REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
}
```

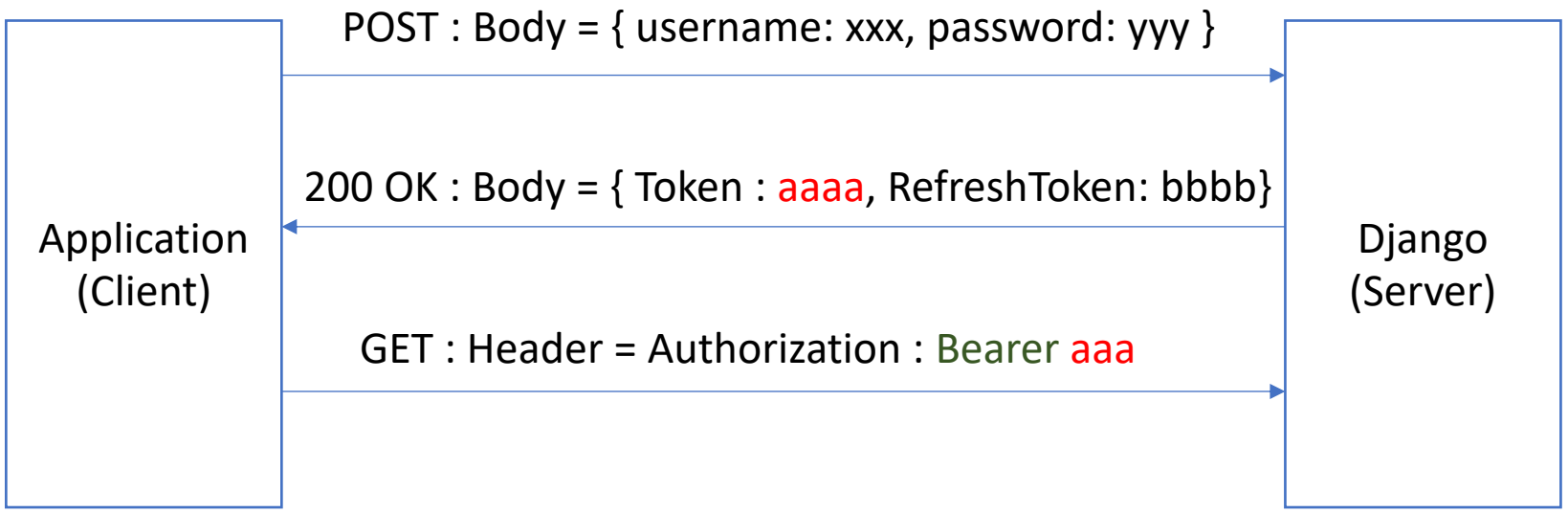
- ทดลองใช้ Chrome เข้า URL account/transaction-viewsets/
- ทดลอง login หน้า Django Admin แล้วกลับมา refresh หน้า account/transaction-viewsets/ ใหม่

Authorization & Authentication (3)

- การทำ Authentication ในตัวอย่างก่อนหน้านี้เป็นการใช้ HTTP Session จากหน้า Admin
- เราจะ Login อย่างไร? เมื่อ client ไม่ได้เปิดหน้า admin

Authorization & Authentication (JWT)

- JSON Web Token (JWT) ใช้สำหรับยืนยันตัวตน client โดยผ่าน Token



Authorization & Authentication (JWT)

- ติดตั้ง Django RestFramework JWT Package

```
pip install djangorestframework-jwt
```

- แก้ไขไฟล์ urls.py เพื่อเพิ่ม URL สำหรับการทำ authentication

urls.py

```
from rest_framework_jwt.views import obtain_jwt_token
#...

urlpatterns = [
    # ...,
    path('api-token-auth/', obtain_jwt_token),
]
```

Authorization & Authentication (JWT) (2) – T013

- แก้ไขไฟล์ settings.py ให้เรียกใช้งาน JWT และตั้งค่า Authorization Header ให้ใช้ Bearer

settings.py

```
import datetime

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_jwt.authentication.JSONWebTokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.BasicAuthentication',
    ),
}

JWT_AUTH = {
    'JWT_AUTH_HEADER_PREFIX': 'Bearer',
    'JWT_EXPIRATION_DELTA': datetime.timedelta(days=300),
}
```

Authorization & Authentication (JWT) (3)

- ทดสอบยิง API ด้วย python requests package

```
pip install requests
```

```
import requests
requests.get('http://localhost:8000/account/customer/')
```

```
login_body = {'username': '<username>', 'password': '<password>'}
resp = requests.post('http://localhost:8000/api-token-auth/', data=login_body)
resp.status_code
resp.json()
```

```
token = resp.json()['token']
header = {'Authorization': f'Bearer {token}'}
requests.get('http://localhost:8000/account/customer', headers=header).text
```

Authorization & Authentication (JWT) (4)

- การตั้งค่า JWT Token Timeout เป็นเวลานาน เป็นเรื่องที่ไม่ควรทำเป็นอย่างยิ่ง
- ควรตั้งค่า Timeout ให้มีขนาดสั้น และใช้การ Refresh Token ช่วย
- วิธีการเปิด Refresh Token

settings.py

```
import datetime

JWT_AUTH = {
    'JWT_AUTH_HEADER_PREFIX': 'Bearer',
    'JWT_ALLOW_REFRESH': True,
}
```

urls.py

```
from rest_framework_jwt.views import refresh_jwt_token
#  ...

urlpatterns = [
    #  ...
    path('api-token-refresh/', refresh_jwt_token),
]
```

Test Your API

สังเกต models.py

- Class Customer จะมี current_amount ที่จะเปลี่ยนค่า ตามผลรวมของ Transaction ทั้งหมด
- เราสามารถแก้ไขให้โปรแกรมคำนวณ current_amount ได้ โดยปรับปรุง account/models.py ได้ดังต่อไปนี้

ปรับปรุง models.py – T014

account/models.py

```
class Customer(models.Model):
     # ...
     def re_calculate_current_amount(self):
        self.current_amount = 0
        for t in self.transaction_set.all():
            if t.type == Transaction.Type.INCOME:
                self.current_amount += t.amount
            else:
                self.current_amount -= t.amount
        self.save()


class Transaction(models.Model):
     #  ...
     def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        self.customer.re_calculate_current_amount()
```

ติดตั้ง Test Components

- Factory Boy ใช้สำหรับจำลอง model objects
- PyTest เป็น library สำหรับใช้ในการ test แทนการใช้คำสั่ง python manage.py test โดยมีข้อดีกว่าดังต่อไปนี้
  - การเขียน Test Function สามารถทำได้ง่ายกว่า
  - สามารถใช้ fixtures (pre-load data) ในการสร้าง test ได้ เช่น ข้อมูล จังหวัด, อำเภอ, เพศ เป็นต้น

```
pip install factory-boy
pip install pytest-django
```

pytest.ini

```
[pytest]
DJANGO_SETTINGS_MODULE = personal_finance.settings
python_files = tests.py test_*.py *_tests.py
addopts = --reuse-db
```

สร้าง Factory Class

account/factories.py

```
import factory
from django.utils import timezone
from django.contrib.auth import get_user_model
from account.models import Customer, Transaction


class UserFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = get_user_model()

    username = factory.Faker('user_name')


class CustomerFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = Customer

    user = factory.SubFactory(UserFactory)
    current_amount = 0
```

สร้าง Factory Class (2)

account/factories.py

```
class TransactionFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = Transaction

    created = factory.Faker('past_datetime', tzinfo=timezone.get_current_timezone())
    amount = factory.Faker('pyint', min_value=0)
    note = ''
    type = factory.Iterator([x[0] for x in Transaction.Type.choices])
    customer = factory.SubFactory(CustomerFactory)
```

- รายละเอียดของ Faker เพิ่มเติมสามารถดูได้จาก https://faker.readthedocs.io/en/master/providers.html

เริ่มเขียน test case และทดสอบ – T015

account/tests.py

```
from django.test import TestCase
from account.models import Transaction
from account.factories import CustomerFactory, TransactionFactory

class TestCustomerCurrentAmount(TestCase):
    def setUp(self):
        self.customer = CustomerFactory()

    def test_transactions(self):
        expected_amount = 0
        for _ in range(0, 10):
            transaction = TransactionFactory(customer=self.customer)
            if transaction.type == Transaction.Type.INCOME:
                expected_amount += transaction.amount
            else:
                expected_amount -= transaction.amount

        self.customer.refresh_from_db()
        self.assertEqual(self.customer.current_amount, expected_amount)
```

- เริ่มทดสอบด้วยคำสั่ง pytest

เริ่มเขียน test case และทดสอบ (2) – T016

account/tests.py

```
#  ...
import datetime
from rest_framework.test import APIClient
from rest_framework import status
from django.urls import reverse

class TestTransactionAPI(TestCase):
    def setUp(self):
        self.customer = CustomerFactory()
        self.client = APIClient()
        self.client.force_authenticate(user=self.customer.user)

    def test_create_transaction_api(self):
        data = {
            'created': datetime.datetime.now(),
            'amount': 1000,
            'note': 'test note',
            'type': Transaction.Type.INCOME,
            'customer': self.customer.id
        }
        response = self.client.post(reverse('transaction-list'), data=data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED, response.data)
        self.assertEqual(Transaction.objects.filter(customer=self.customer).count(), 1)
        self.customer.refresh_from_db()
        self.assertEqual(self.customer.current_amount, 1000)
```

3rd Party Packages

API Documentation

- Django Rest Framework สามารถ generate document ได้หลากหลายวิธี หนึ่งในนั้นคือการใช้ package drf-yasg

```
pip install drf-yasg
```

settings.py

```
INSTALLED_APPS = [
    ...
    'drf_yasg',
]
```

API Documentation (2) – T017

urls.py

```
from rest_framework import permissions
from drf_yasg.views import get_schema_view
from drf_yasg import openapi

schema_view = get_schema_view(
    openapi.Info(
        title="Personal Finance API",
        default_version='v1',
        description="bla bla..",
    ),
    public=True,
    permission_classes=[permissions.AllowAny],
)

urlpatterns = [
    path('swagger/', schema_view.with_ui('swagger', cache_timeout=0)),
]
```

- ทดสอบโดยการเปิด browser ไปที่ http://localhost:8000/swagger/

Django-Filter

- Django Filter เป็น 3rd party package ที่นิยมนำมาใช้งานร่วมกับ Django Rest API
- สำหรับเพิ่มความสามารถในการ Filter ค่าที่ต้องการจาก URL โดยอัตโนมัติ

```
pip install django-filter
```

settings.py

```
INSTALLED_APPS = [
    ...
    'django_filters',
]

REST_FRAMEWORK = {
    'DEFAULT_FILTER_BACKENDS': (
        'django_filters.rest_framework.DjangoFilterBackend',
    ),
    ...
}
```

Django-Filter (2) – T018

account/views.py

```
from rest_framework import viewsets


class CustomerViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Customer.objects.all()
    serializer_class = CustomerSerializer
    filterset_fields = ('user__username',)


class TransactionViewSet(viewsets.ModelViewSet):
    queryset = Transaction.objects.all()
    serializer_class = TransactionSerializer
    filterset_fields = ('customer__id', 'customer__user__username')
```

- http://localhost:8000/account/customer-viewsets/?user__username=admin
- http://localhost:8000/account/transaction-viewsets/?customer__id=1

Django Extensions – T019

- Django Extensions เป็นการเพิ่ม feature ต่าง ๆ ให้การพัฒนา Django ทำได้อย่างง่ายยิ่งขึ้น

```
pip install django-extensions
pip install ipython
```

settings.py

```
INSTALLED_APPS = [
    ...
    'django_extensions',
]

SHELL_PLUS = "ipython"
```

ทดสอบรันคำสั่ง

```
python manage.py show_urls
python manage.py shell_plus
```

CORS Header

ติดตั้ง Package django-cors-headers – T020

```
pip install django-cors-headers
```

settings.py

```
INSTALLED_APPS = [
  # ...
  'corsheaders',
]

MIDDLEWARE = [
  # ...
  'corsheaders.middleware.CorsMiddleware',
  'django.middleware.common.CommonMiddleware',
]

CORS_ALLOW_ALL_ORIGINS = DEBUG
```