

# Django REST API Workshop

A Workshop on Django web application development

<https://github.com/WasinTh/covid-self-monitoring-backend>



# ทบทวนความรู้พื้นฐาน Django

- สร้าง Django Project ชื่อ “covid\_self\_monitoring”

```
django-admin startproject covid_self_monitoring
```

- สร้าง Django Application ชื่อ “monitor”

```
python manage.py startapp monitor
```



# สร้างฐานข้อมูล

monitor/models.py

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

User = get_user_model()

class Symptom(models.Model):
    """อาการผิดปกติ"""
    name = models.CharField(max_length=1024)

    def __str__(self):
        return self.name
```



## สร้างฐานข้อมูล (2)

monitor/models.py

```
class Measurement(models.Model):
    """การวัดค่า"""
    created = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    temperature = models.DecimalField(max_digits=4, decimal_places=2, help_text='อุณหภูมิร่างกาย')
    o2sat = models.IntegerField(help_text='ออกซิเจนในเลือด')
    systolic = models.IntegerField(help_text='ความดันตัวบน')
    diastolic = models.IntegerField(help_text='ความดันตัวล่าง')
    symptoms = models.ManyToManyField(Symptom, blank=True, help_text='อาการที่พบ')

    @property
    def symptoms_display(self):
        return ', '.join(self.symptoms.values_list('name', flat=True))

    class Meta:
        ordering = ['-created']
```



# สร้าง Admin Page – T001

monitor/admin.py

```
from django.contrib import admin
from monitor import models

@admin.register(models.Symptom)
class SymptomAdmin(admin.ModelAdmin):
    pass

@admin.register(models.Measurement)
class MeasurementAdmin(admin.ModelAdmin):
    list_display = ['user', 'created', 'temperature',
                   'o2sat', 'systolic', 'diastolic', 'symptoms_display']
    list_filter = ['user', 'created']
    filter_horizontal = ['symptoms']
```



# ทบทวนความรู้พื้นฐาน Django

- แก้ไขไฟล์ settings.py เพื่อเพิ่ม account เข้าไปใน INSTALLED\_APPS
- MakeMigrations และ Migrate Database
- สร้าง superuser account

```
python manage.py createsuperuser
```

- รัน Django Project

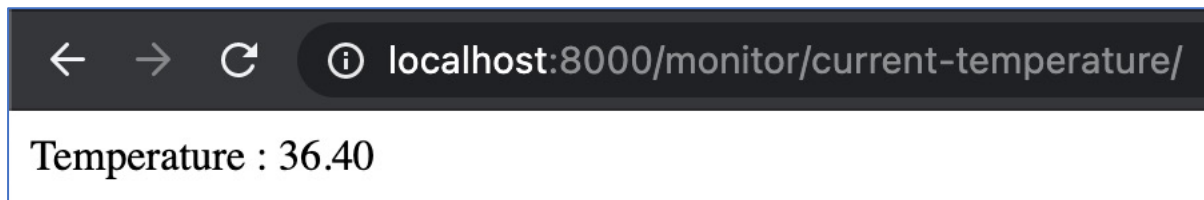
```
python manage.py runserver 0.0.0.0:8000
```

- เปิด browser ไปที่ URL : <http://localhost:8000/admin/>
  - สร้าง Symptom เช่น ไอ, เสมหะ, น้ำมูก, ท้องเสีย, ไม่รับรส
  - สร้าง Measurement



## ทบทวนความรู้พื้นฐาน Django (3) – T002

- สร้าง URL ชื่อ /monitor/current-temperature/
  - สำหรับแสดงรายละเอียดของ measurement ล่าสุดของ user ID ที่ได้รับมา



1. สร้าง function ใหม่ในไฟล์ `monitor/views.py` ให้ return ตัวเลขอุณหภูมิปัจจุบันออกมา  

```
from django.http import HttpResponse
```
2. สร้างไฟล์ `monitor/urls.py` โดยเพิ่ม URL ชื่อ `current-temperature`
3. แก้ไขไฟล์ `covid_self_monitoring/urls.py` ให้ include URL ของ `monitor` ทั้งหมดลงไป
4. ทดสอบเรียกใช้งาน URL นี้ผ่าน web browser

# Django REST API



# 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

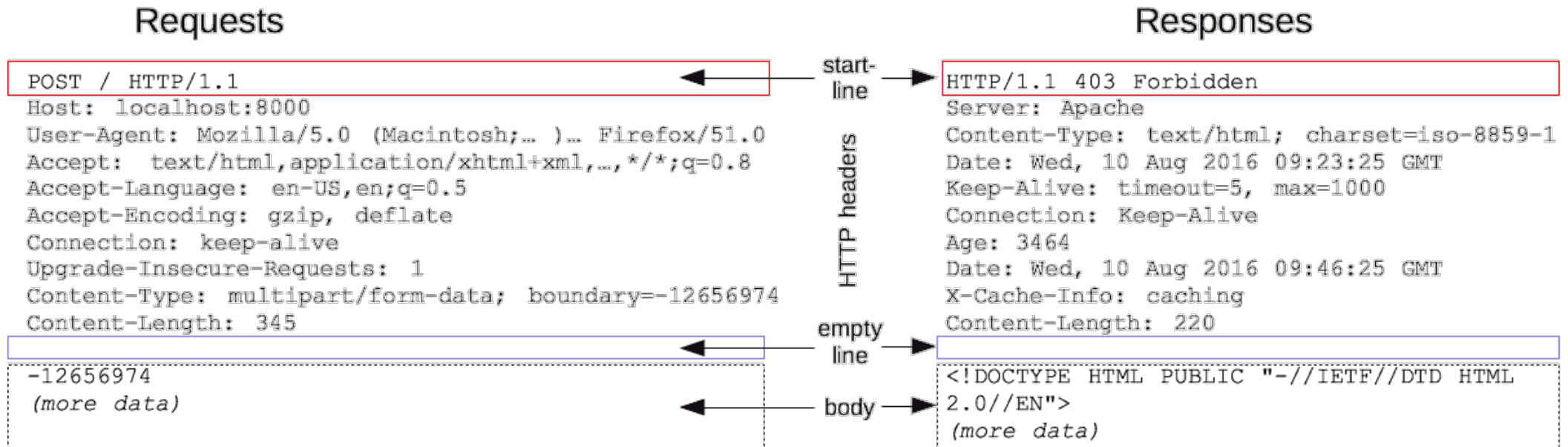
- Back-End

- Django
- Node.js
- PHP
- .NET

- Front-End

- React
- Vue
- Angular
- Flutter

# HTTP Message

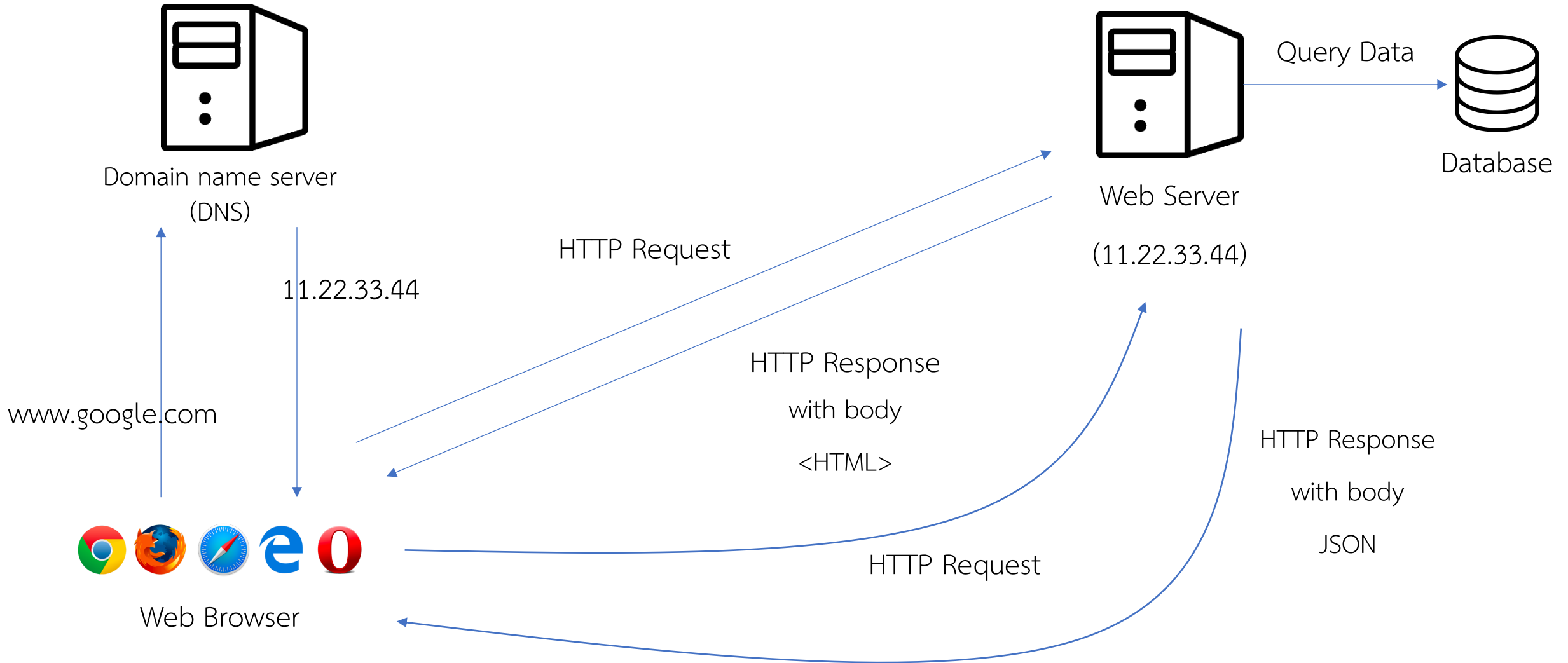


๒

## 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

monitor/views.py

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

def current_temperature(request):
    data = {'temperature': str(Measurement.objects.last().temperature)}
    return HttpResponse(json.dumps(data))
```



# List Measurement REST API – T004

- ใช้ Django Admin สร้าง Measurement เพิ่ม
- สร้าง URL ใหม่สำหรับเรียกแสดง List ของ Measurement ทั้งหมด
- Example URL
  - <http://localhost:8000/monitor/all-measurement/>

```
localhost:8000/monitor/all-measurement/

[{"created": "2021-05-13 06:49:04.679771+00:00", "user_id": 1, "temperature": "36.40", "o2sat": 99, "systolic": 120, "diastolic": 80, "symptoms": "\u0e44\u0e2d, \u0e21\u0e35\u0e40\u0e2a\u0e21\u0e2b\u0e30"}, {"created": "2021-05-13 07:53:11.711103+00:00", "user_id": 1, "temperature": "36.80", "o2sat": 95, "systolic": 122, "diastolic": 84, "symptoms": "\u0e44\u0e2d"}]
```

# 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 ทำให้มี learning curve สูง



# 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

monitor/serializers.py

```
import datetime
from rest_framework import serializers

class MeasurementSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    created = serializers.DateTimeField(default=datetime.datetime.now())
    temperature = serializers.DecimalField(max_digits=4, decimal_places=2)
    o2sat = serializers.IntegerField()
    systolic = serializers.IntegerField()
    diastolic = serializers.IntegerField()
```



## ติดตั้ง Package เสริม

- package Django Extensions และ iPython จะช่วยให้การใช้ Django Shell สามารถทำได้ง่ายขึ้น

```
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
```



## ทำความรู้จัก Serializer (2)

- รัน python manage.py shell แล้วทดสอบคำสั่งต่อไปนี้

```
from monitor.serializers import MeasurementSerializer
m = Measurement.objects.last()
serializer = MeasurementSerializer(m)
print(serializer.data)
```

- เราสามารถ serialize queryset ได้เช่นกัน

```
serializer = MeasurementSerializer(Measurement.objects.all(), many=True)
print(serializer.data)
```



# ปรับปรุง Serializer (เพิ่ม User Field) – T006

monitor/serializers.py

```
import datetime
from rest_framework import serializers
from django.contrib.auth import get_user_model

User = get_user_model()

class MeasurementSerializer(serializers.Serializer):
    id = serializers.IntegerField(read_only=True)
    created = serializers.DateTimeField(default=datetime.datetime.now())
    temperature = serializers.DecimalField(max_digits=4, decimal_places=2)
    o2sat = serializers.IntegerField()
    systolic = serializers.IntegerField()
    diastolic = serializers.IntegerField()
    user = serializers.PrimaryKeyRelatedField(queryset=User.objects.all())
```



## ทำความรู้จัก Serializer (3)

```
from monitor.serializers import MeasurementSerializer
serializer = MeasurementSerializer(Measurement.objects.last())
print(serializer.data)
```



## ทำความรู้จัก Serializer (4) – T007

- Serializer สามารถใช้ในการสร้าง object จาก JSON ได้

```
from monitor.serializers import MeasurementSerializer
data={'temperature':36.5, 'o2sat':98, 'systolic': 120, 'diastolic': 79,
'user': User.objects.last().id}
serializer = MeasurementSerializer(data=data)
serializer.is_valid()
serializer.save()
```

monitor/serializers.py

```
from monitor.models import Measurement

class MeasurementSerializer(serializers.Serializer):
    # .....

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



# ทดลองใช้ ModelSerializer

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

monitor/serializers.py

```
import datetime
from rest_framework import serializers
from monitor.models import Measurement

class MeasurementSerializer(serializers.ModelSerializer):
    created = serializers.DateTimeField(default=datetime.datetime.now())

    class Meta:
        model = Measurement
        fields = ['id', 'created', 'temperature', 'o2sat', 'systolic', 'diastolic', 'user']
```



# Nested Serializer – T008

monitor/serializers.py

```
from monitor.models import Measurement, Symptom

class SymptomSerializer(serializers.ModelSerializer):
    class Meta:
        model = Symptom
        fields = '__all__'

class MeasurementSerializer(serializers.ModelSerializer):
    created = serializers.DateTimeField(default=datetime.datetime.now())
    symptoms = SymptomSerializer(many=True)

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

```
from monitor.serializers import MeasurementSerializer
serializer = MeasurementSerializer(Measurement.objects.last())
print(serializer.data)
```





# นำ Serializer มาใช้กับ Views - T009

monitor/views.py

```
from rest_framework.response import Response
from rest_framework.decorators import api_view
from monitor.models import Measurement
from monitor.serializers import MeasurementSerializer

@api_view([ 'GET' ])
def all_measurement(request):
    serializer = MeasurementSerializer(Measurement.objects.all(), many=True)
    return Response(data=serializer.data)
```

# การใช้งาน Django REST framework

- Function Based Views จากตัวอย่างก่อนหน้านี้ 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 – T010

monitor/views.py

```
from rest_framework.views import APIView

class AllMeasurementView(APIView):
    def get(self, request):
        serializer = MeasurementSerializer(Measurement.objects.all(), many=True)
        return Response(data=serializer.data)
```

monitor/urls.py

```
from monitor import views

urlpatterns = [
    # .....
    path('all-measurement-api-view/', views.AllMeasurementView.as_view()),
]
```

# การใช้งาน Django REST framework

- Function Based Views จากตัวอย่างก่อนหน้านี้ 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 สำหรับการ update ค่าของ object
  - DestroyAPIView รับ DELETE สำหรับการ 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

monitor/views.py

```
from rest_framework import generics

class MeasurementGenericsView(generics.ListAPIView):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer
```

monitor/urls.py

```
from monitor import views

urlpatterns = [
    # .....
    path('measurement-generics-view/', views.MeasurementGenericsView.as_view()),
]
```



## Generics API View (3) – T011

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

monitor/views.py

```
from rest_framework import generics

class MeasurementGenericsView(generics.ListCreateAPIView):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer
```





# สร้าง REST API สำหรับจัดการ Symptom – T012

- เมื่อเรียก URL `/monitor/symptom-generics-view/` จะต้องสามารถเห็น list ของ symptom และสร้าง symptom ใหม่ได้
- แก้ไขไฟล์ `serializers.py`, `views.py`, และ `urls.py`
- Hint : ใช้ `generics.ListCreateAPIView`

# การใช้งาน Django REST framework

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

# ViewSet & 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 เพื่อลดความซ้ำซ้อนของไฟล์



# ViewSets & Router – ทดลองสร้าง ViewSets – T013

monitor/views.py

```
from rest_framework import viewsets

class MeasurementViewsets(viewsets.ModelViewSet):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer

class SymptomViewsets(viewsets.ReadOnlyModelViewSet):
    queryset = Symptom.objects.all()
    serializer_class = SymptomSerializer
```

monitor/urls.py

```
from django.urls import include
from rest_framework.routers import DefaultRouter
from monitor import views

router = DefaultRouter()
router.register('measurement-viewsets', views.MeasurementViewsets)
router.register('symptom-viewsets', views.SymptomViewsets)

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

- ระบบปัจจุบันอนุญาตให้ใครก็ได้สามารถเข้ามายัง API ได้ เราสามารถเพิ่ม Permission ในการเข้าถึง API ได้ ดังต่อไปนี้
- เพิ่ม code ใน views.py แล้วทดสอบ /monitor/ measurement-generics-view/ ใน Chrome Incognito Mode

monitor/views.py

```
from rest_framework import permissions

class MeasurementGenericsView(generics.ListCreateAPIView):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]
```

# Authorization & Authentication (2) – T014

- กรณีต้องการตั้งค่า Permission เหมือนกันทั้ง project สามารถตั้งค่าได้ผ่านทาง settings.py
- ทดสอบเพิ่ม code นี้ลงไปใน settings.py

| settings.py                                                                                                                                      |
|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>REST_FRAMEWORK = {<br/>    'DEFAULT_PERMISSION_CLASSES': (<br/>        'rest_framework.permissions.IsAuthenticated',<br/>    ),<br/>}</pre> |

- ทดลองใช้ Chrome Incognito mode เข้า URL monitor/measurement-viewssets/
- ทดลองเปิด tab ใหม่ไปหน้า Admin จากนั้น login หน้า Django Admin แล้วกลับมา refresh หน้าเว็บไซต์ใหม่อีกครั้ง

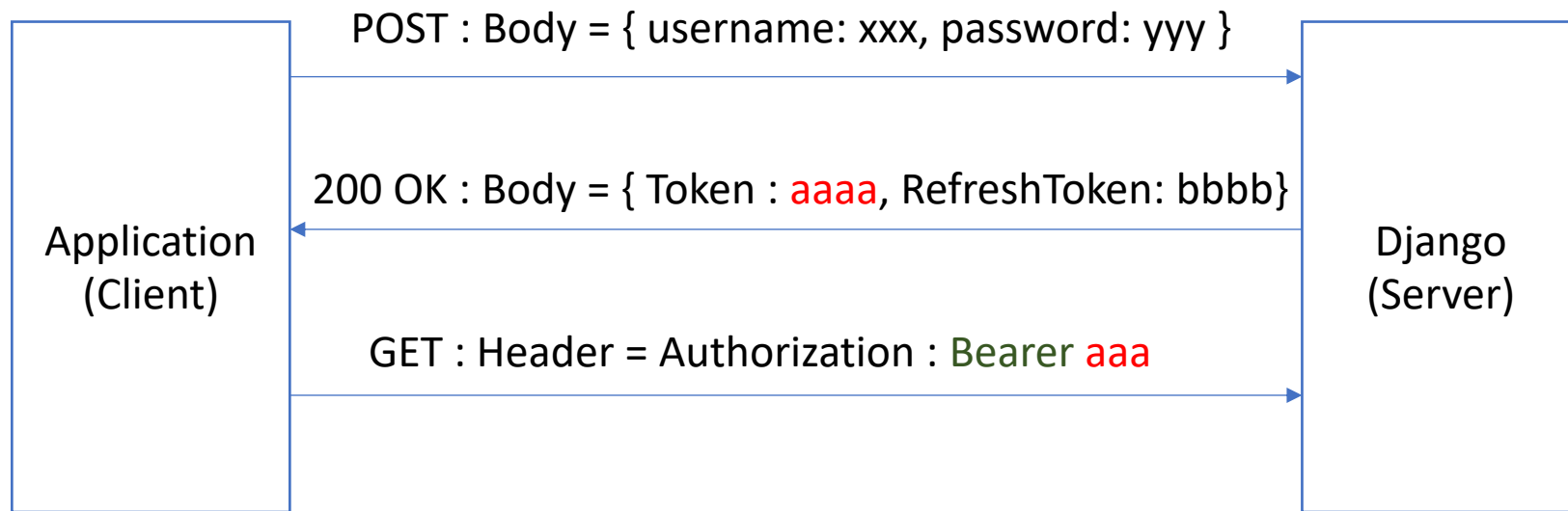


# Authorization & Authentication (3)

- การทำ Authentication ในตัวอย่างก่อนหน้านี้เป็นการใช้ HTTP Session จากหน้า Admin
- หากต้องการให้ User ที่ไม่ใช่ Admin เข้าใช้ API เหล่านี้ จะทำอย่างไร?

# Authorization & Authentication (JWT)

- JSON Web Token (JWT) ใช้สำหรับยืนยันตัวตน client โดยผ่าน Token





# Authorization & Authentication (JWT)

- ติดตั้ง Django RestFramework JWT Package

```
pip install djangorestframework_simplejwt
```

- แก้ไขไฟล์ covid\_self\_monitoring/urls.py เพื่อเพิ่ม URL สำหรับการทำ authentication

covid\_self\_monitoring/urls.py

```
from rest_framework_simplejwt import views as jwt_views
#...

urlpatterns = [
    # ...,
    path('api-token-auth/', jwt_views.TokenObtainPairView.as_view()),
    path('api-token-refresh/', jwt_views.TokenRefreshView.as_view()),
]
```



## Authorization & Authentication (JWT) (2) – T015

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

settings.py

```
from datetime import timedelta

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

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(hours=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),
}
```



## Authorization & Authentication (JWT) (3)

- ทดสอบยิง API ด้วย python requests package

```
pip install requests
```

```
import requests  
requests.get('http://localhost:8000/monitor/symptom-viewsets/')
```

```
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()['access']  
header = {'Authorization': f'Bearer {token}'}  
requests.get('http://localhost:8000/monitor/symptom-viewsets/', headers=header).text
```

# Authorization & Authentication (JWT) (4)

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

settings.py

```
SIMPLE_JWT = {  
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),  
    'REFRESH_TOKEN_LIFETIME': timedelta(days=30),  
}
```

Test Your API



# ติดตั้ง Test Components

- Factory Boy ใช้สำหรับจำลอง model objects
- PyTest เป็น library สำหรับใช้ในการ test แทนการใช้คำสั่ง python manage.py test โดยมีข้อดีกว่าดังต่อไปนี้
  - การเขียน Test Function สามารถทำได้ง่ายขึ้น
  - 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 = covid_self_monitoring.settings
python_files = tests.py test_*.py *_tests.py
addopts = --reuse-db
```





# สร้าง Factory Class

monitor/factories.py

```
import factory
from django.utils import timezone
from django.contrib.auth import get_user_model
from monitor import models

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

        username = factory.Faker('user_name')

class SymptomFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = models.Symptom

        name = factory.Sequence(lambda n: f"อาการ {n}")
```



## สร้าง Factory Class (2)

monitor/factories.py

```
class MeasurementFactory(factory.django.DjangoModelFactory):
    class Meta:
        model = models.Measurement

    created = factory.Faker('past_datetime', tzinfo=timezone.get_current_timezone())
    user = factory.SubFactory(UserFactory)
    temperature = factory.Faker('pydecimal', min_value=36, max_value=40)
    o2sat = factory.Faker('pyint', min_value=90, max_value=100)
    systolic = factory.Faker('pyint', min_value=120, max_value=130)
    diastolic = factory.Faker('pyint', min_value=90, max_value=100)

    @factory.post_generation
    def symptoms(self, create, extracted, **kwargs):
        if not create:
            # Simple build, do nothing.
            return

        if extracted:
            # A list of groups were passed in, use them
            for symptom in extracted:
                self.symptoms.add(symptom)
```

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



# เริ่มเขียน test case และทดสอบ – T016

monitor/tests.py

```
from django.test import TestCase
from monitor.models import Measurement
from monitor.factories import UserFactory, SymptomFactory, MeasurementFactory

class TestCreateMeasurement(TestCase):
    def setUp(self):
        self.user_1 = UserFactory()
        self.user_2 = UserFactory()
        self.symptoms = [SymptomFactory() for _ in range(0, 3)]

    def test_create_single_user(self):
        for _ in range(0, 10):
            MeasurementFactory(user=self.user_1, symptoms=self.symptoms)

        self.assertEqual(10, Measurement.objects.count())

    def test_create_multiple_user(self):
        for _ in range(0, 10):
            MeasurementFactory(user=self.user_1, symptoms=self.symptoms)

        for _ in range(0, 5):
            MeasurementFactory(user=self.user_2)

        self.assertEqual(15, Measurement.objects.count())
        self.assertEqual(10, Measurement.objects.filter(user=self.user_1).count())
        self.assertEqual(5, Measurement.objects.filter(user=self.user_2).count())
```

- เริ่มทดสอบด้วยคำสั่ง pytest



## ปรับปรุง Serializer ด้วย user field

- Field User ไม่จำเป็นต้องให้ Front-End ระบุเจาะจงเข้ามา สามารถใช้ข้อมูล user จาก Authentication session ได้ทันที

monitor/serializers.py

```
class MeasurementSerializer(serializers.ModelSerializer):
    created = serializers.DateTimeField(default=datetime.datetime.now())
    symptoms = SymptomSerializer(many=True)
    user = serializers.PrimaryKeyRelatedField(queryset=User.objects.all(),
                                              default=serializers.CurrentUserDefault())

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



# เริ่มเขียน test case และทดสอบ (2) – T017

monitor/tests.py

```
from django.utils import timezone
from django.urls import reverse
from django.forms.models import model_to_dict
from rest_framework import status
from rest_framework.test import APIClient

class TestCreateMeasurementAPI(TestCase):
    def setUp(self):
        self.user = UserFactory()
        self.client = APIClient()
        self.client.force_authenticate(user=self.user)
        self.symptoms = [SymptomFactory() for _ in range(0, 3)]
        self.data = {
            'created': timezone.now(),
            'temperature': 36.8,
            'o2sat': 97,
            'systolic': 124,
            'diastolic': 90,
            'symptoms': [model_to_dict(s) for s in self.symptoms]
        }

    def test_create_measurement_api(self):
        response = self.client.post(reverse('measurement-list'), data=self.data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED, response.data)
        self.assertEqual(Measurement.objects.filter(user=self.user).count(), 1)
```



# Django Serializer และ Manytomany Field

- จากการ Test Code ก่อนหน้านี้จะเห็นว่าระบบแสดง Error เนื่องจาก MeasurementSerializer ไม่สามารถรับค่า symptoms ได้

monitor/serializers.py

```
class SymptomSerializer(serializers.ModelSerializer):  
    id = serializers.IntegerField()  
  
    class Meta:  
        model = Symptom  
        fields = '__all__'
```



# Django Serializer และ Manytomany Field (2) – T018

monitor/serializers.py

```
class MeasurementSerializer(serializers.ModelSerializer):

    #.....

    def create(self, validated_data):
        symptoms = validated_data.pop('symptoms')
        measurement = Measurement.objects.create(**validated_data)
        measurement.symptoms.set(Symptom.objects.filter(id__in=[s['id'] for s in symptoms]))
        return measurement

    def update(self, instance, validated_data):
        symptoms = validated_data.pop('symptoms')
        measurement = super().update(instance, validated_data)
        measurement.symptoms.set(Symptom.objects.filter(id__in=[s['id'] for s in symptoms]))
        return measurement
```

# Django Rest Framework Validation



# การ Validate ค่าของ Django Rest Framework

- การ Validate โดยปกติจะทำในส่วนของ Serializer
  - เนื่องจาก Code มีโอกาสถูกนำไป re-use ใช้งานได้ง่ายกว่าใน view
- สามารถทำได้ 3 แบบคือ
  - ใช้ Build in Validator เช่น UniqueValidator โดยส่งผ่านค่าผ่านทาง property validators
    - ไม่นิยมใช้เนื่องจากมี class ค่อนข้างจำกัด
  - เขียน function ชื่อ `validate_<field name>(self, data)` ไว้ภายใน serializer
    - ใช้สำหรับการ validate ค่าของ field เพียงหนึ่งค่า
  - override function `validate(self, attrs)` ไว้ภายใน serializer
    - ใช้เมื่อต้องการนำข้อมูลจากหลายๆ field มารวมกันเพื่อ validate



## สร้าง Validate Temperature – T019

- ทดลองสร้าง Validator โดยจะ return ค่า 400 Error กลับไปเมื่อการส่ง Temperature ที่มีค่าน้อยกว่า 30 หรือ มากกว่า 50

monitor/serializers.py

```
class MeasurementSerializer(serializers.ModelSerializer):  
    #.....  
  
    def validate_temperature(self, data):  
        if data < 30 or data > 50:  
            raise serializers.ValidationError('Unreasonable temperature')  
        return data
```

monitor/tests.py

```
class TestCreateMeasurementAPI(TestCase):  
    # .....  
  
    def test_error_temperature(self):  
        self.data['temperature'] = 90  
        response = self.client.post(reverse('measurement-list'), data=self.data, format='json')  
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST, response.data)
```



## Validate ผู้ใช้งาน

- Validate ว่า user\_1 ไม่สามารถสร้างข้อมูล measurement ของ user\_2 ได้

monitor/serializers.py

```
class MeasurementSerializer(serializers.ModelSerializer):  
    # .....  
  
    def validate_user(self, data):  
        if self.instance and self.instance.user != data:  
            raise serializers.ValidationError('Cannot update other user instance')  
        return data
```



## Validate ผู้ใช้งาน (2) – T020

monitor/tests.py

```
class TestUpdateMeasurementAPI(TestCase):
```

```
    def setUp(self):
```

```
        self.user_1 = UserFactory()
```

```
        self.user_2 = UserFactory()
```

```
        self.data = {
```

```
            'created': timezone.now(),
```

```
            'temperature': 36.8,
```

```
            'o2sat': 97,
```

```
            'systolic': 124,
```

```
            'diastolic': 90,
```

```
            'symptoms': []
```

```
        }
```

```
    def test_update_error(self):
```

```
        client = APIClient()
```

```
        client.force_authenticate(user=self.user_2)
```

```
        measurement = MeasurementFactory(user=self.user_1)
```

```
        response = client.put(
```

```
            reverse('measurement-detail', kwargs={'pk': measurement.id}),
```

```
            data=self.data,
```

```
            format='json'
```

```
        )
```

```
        self.assertEqual(response.status_code, status.HTTP_400_BAD_REQUEST, response.data)
```

รัน Test ด้วยคำสั่ง

```
pytest monitor/tests.py::TestUpdateMeasurementAPI
```

# 3rd Party Packages



# API Documentation

- Django Rest Framework สามารถ generate document ได้หลากหลายวิธี หนึ่งในนั้นคือการ  
ใช้ package drf-yasg

```
pip install drf-yasg
```

```
settings.py
```

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



## API Documentation (2) – T021

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="Covid Self Monitoring 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 3<sup>rd</sup> 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) – T022

account/views.py

```
from rest_framework import viewsets

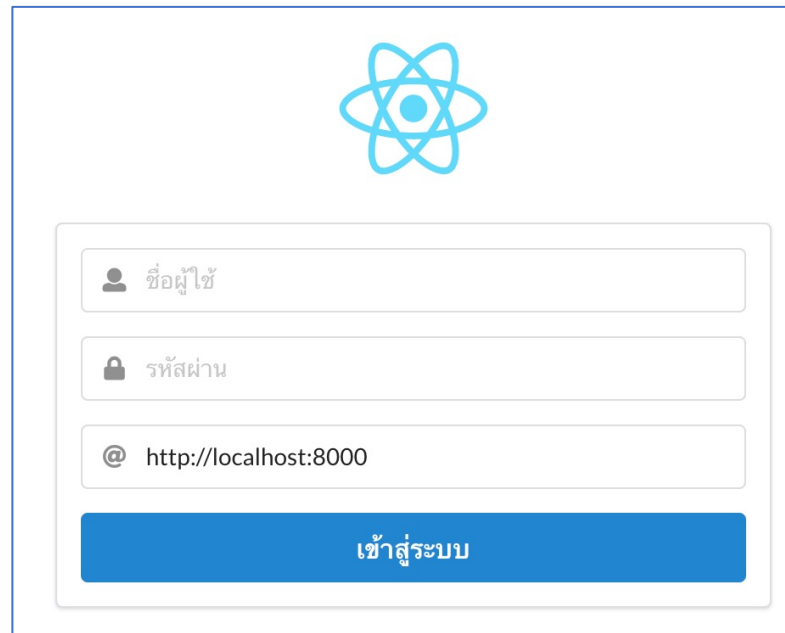
class MeasurementViewsets(viewsets.ModelViewSet):
    queryset = Measurement.objects.all()
    serializer_class = MeasurementSerializer
    filterset_fields = ('user__id', 'user__username')
```

- [http://localhost:8000/monitor/measurement-viewsets/?user\\_\\_id=2](http://localhost:8000/monitor/measurement-viewsets/?user__id=2)
- [http://localhost:8000/monitor/measurement-viewsets/?user\\_\\_username=admin](http://localhost:8000/monitor/measurement-viewsets/?user__username=admin)

# Front-End Integration

- เปิด Browser ไปที่ URL ต่อไปนี้เพื่อรัน Front-End Application

<https://selfmonitoring.tailor-solutions.com/>



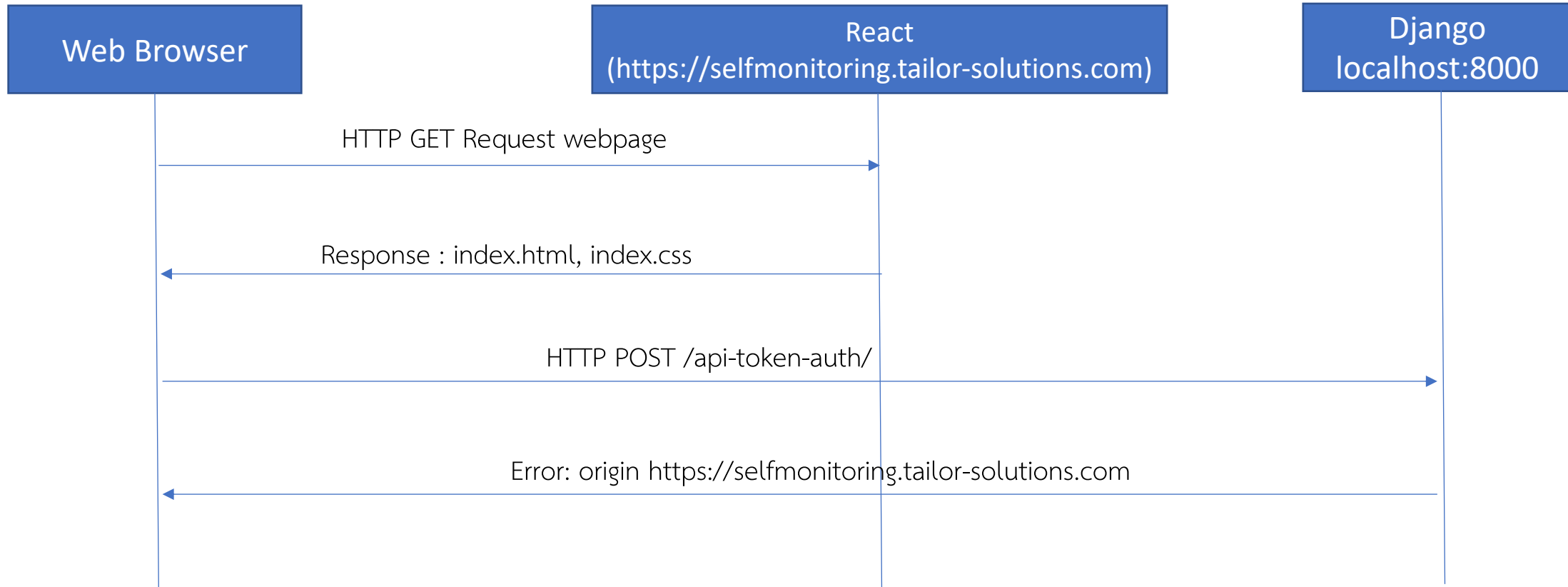
The image shows a web application interface for login. At the top center is the React logo (a blue atom-like symbol). Below it is a white rectangular box containing three input fields and a button. The first input field has a user icon and the label 'ชื่อผู้ใช้' (Username). The second input field has a lock icon and the label 'รหัสผ่าน' (Password). The third input field has an '@' icon and the text 'http://localhost:8000'. Below these fields is a blue button with the white text 'เข้าสู่ระบบ' (Login).

(Source Code : <https://github.com/WasinTh/covid-self-monitoring-frontend>)

# CORS Header

# Cross-Origin Resource Sharing (CORS)

```
✖ Access to XMLHttpRequest at 'http://localhost:8000/api-token-auth/' from origin 'https://selfmonitoring.tailor-solutions.com' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource. /login:1
Error: Network Error
    at t.exports (createError.js:16)
    at XMLHttpRequest.d.onerror (xhr.js:84)
    at LoginScreen.js:40
✖ > POST http://localhost:8000/api-token-auth/ net::ERR_FAILED xhr.js:177
>
```





# ติดตั้ง Package django-cors-headers – T023

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

settings.py

```
INSTALLED_APPS = [  
    # ...  
    'corsheaders',  
]  
  
MIDDLEWARE = [  
    # ...  
    'corsheaders.middleware.CorsMiddleware',  
    'django.middleware.common.CommonMiddleware',  
]  
  
CORS_ALLOW_ALL_ORIGINS = DEBUG  
  
CORS_ALLOWED_ORIGINS = [  
    "https://selfmonitoring.tailor-solutions.com",  
]
```