Server-Side

Midterm: wee4 - week8

▼ Week 2

Topic : Install Django & Views

> If u can’t use pip → python -m pip <command>

```
> python --version

# create folder and open folder in VScode next to set Environ
> mkdir yourproject
> cd yourproject

# Install virtualenv
> pip install virtualenv

# Create a virtual environment
> py -m venv myvenv

# Activate virtual environment
> myvenv\Scripts\activate.bat

# Install Django
> pip install django

# Check version Django
> python -m django --version
```

```
# Create Project in Folder
> django-admin startproject mysite

# Test run manage.py (default port is 8000)
> python manage.py runserver

# Database PostgreSQL
> pip install psycopg2
OR
> pip install psycopg2-binary
OR
> brew install postgresql # for MacOS

# Create app in Project
> python manage.py startapp polls

# In setting.py (default password pgAdmin is password)
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql",
        "NAME": "mypolls",
        "USER": "db_username",
        "PASSWORD": "password",
        "HOST": "localhost",
        "PORT": "5432",
    }
}

# Add app to INSTALLED_APPS
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
```

```
    "django.contrib.staticfiles",
    # Add your apps here
    "polls",
]

# Model get to pgAdmin(Database)
> python manage.py makemigrations polls
> python manage.py migrate polls
```

```
# Playing with API (shell)
> python manage.py shell

# ทำการ import models Question, Choice
>>> from polls.models import Question, Choice

# ทำการ SELECT ข้อมูลในตาราง question (SELECT * FROM question)
>>> Question.objects.all()
<QuerySet []>

# สร้าง instance ของ Question
>>> from django.utils import timezone
>>> q = Question(question_text="What is new?", pub_date=timez

# บันทึกข้อมูลลงใน table question
>>> q.save()

# ID ของ question ที่สร้าง = 1
>>> q.id
1

# ตัวแปร q เป็น instance ของ class Question ดังนั้นจะมี attribute qu
# ซึ่งสามารถเข้าถึงได้โดยใช้ dot notation ของ Python
>>> q.question_text
"What's new?"
>>> q.pub_date
```

```
datetime.datetime(2024, 2, 26, 13, 0, 0, 115217, tzinfo=datet

# แก้ไขข้อมูลเปลี่ยน question_text จาก "What's new?" -> "What's up?"
# SQL: UPDATE question SET question.question_text="What's up?
>>> q.question_text = "What is up?"
# สั่ง save เพื่อบันทึกข้อมูลลง database
>>> q.save()

# Question.objects.all() แสดงข้อมูลทั้งหมดในตาราง question
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>



# In next next week, you can use jupyter
# Learn about views more
```

▼ Week3

Topic: Model & DateTime, Timezone

```
> python manage.py makemigrations
> python manage.py migrate
```

How to write syntax Models ?

```
from django.db import models

class nameClass(models.Model):
    name_field = models.CharField(max_length=150)
    name_field = models.CharField(max_length=150)
    name_field = models.CharField(max_length=150)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

```
# Example
class Author(models.Model):
    first_name = models.CharField(max_length=150)
    last_name = models.CharField(max_length=200)
    email = models.CharField(max_length=150)

    def __str__(self):
        return f"{self.first_name} {self.last_name}"

class Category(models.Model):
    name = models.CharField(max_length=100)

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

Model fields ที่ใช้บ่อยๆ

Fields types

- BooleanField(**options)
- CharField(max_length=None, **options)
- EmailField(max_length=254, **options)
- URLField(max_length=200, **options)
- UUIDField(**options)
- TextField(**options)
- DateField(auto_now=False, auto_now_add=False, **options)
    - auto_now = True คือจะบันทึกค่า datetime.now() ทุกครั้งที่มีการแก้ไขค่า (INSERT + UPDATE)
    - auto_now_add = True คือจะบันทึกค่า datetime.now() ตอนที่สร้างใหม่ (INSERT)
- DateTimeField(auto_now=False, auto_now_add=False, **options)

- TimeField(auto_now=False, auto_now_add=False, **options)
- FileField(upload_to='', storage=None, max_length=100, **options)
  - upload_to คือกำหนด path ที่จะ save file

```
class MyModel(models.Model):
    # file will be uploaded to MEDIA_ROOT/uploads
    upload = models.FileField(upload_to="uploads/")
    # or...
    # file will be saved to MEDIA_ROOT/uploads/2015/01/30
    upload = models.FileField(upload_to="uploads/%Y/%m/%d/
```

- ImageField(upload_to=None, height_field=None, width_field=None, max_length=100, **options)
  - สืบทอด attributes และ methods ทั้งหมดจาก `FileField`
  - ทำการ validate ให้ว่าเป็น object ของ image ที่เหมาะสม และ สามารถกำหนด `height_field` และ `width_field`
- DecimalField(max_digits=None, decimal_places=None, **options)

```
models.DecimalField(max_digits=5, decimal_places=2)
```

- IntegerField(**options)
  - ค่าตั้งแต่ -2147483648 ถึง 2147483647 รองรับใน database ทุกตัวที่ supported โดย Django.
- PositiveIntegerField(**options)
- JSONField(encoder=None, decoder=None, **options)

```
from django.db import models

class Student(models.Model):
    FRESHMAN = "FR"
    SOPHOMORE = "SO"
    JUNIOR = "JR"
```

```
    SENIOR = "SR"
    GRADUATE = "GR"
    YEAR_IN_SCHOOL_CHOICES = {
        FRESHMAN: "Freshman",
        SOPHOMORE: "Sophomore",
        JUNIOR: "Junior",
        SENIOR: "Senior",
        GRADUATE: "Graduate",
    }
    year_in_school = models.CharField(
        max_length=2,
        choices=YEAR_IN_SCHOOL_CHOICES,
        default=FRESHMAN,
    )

    def is_upperclass(self):
        return self.year_in_school in {self.JUNIOR, self.SENI
```

- choices: กำหนด ENUM ให้เลือกเฉพาะค่าที่กำหนด
- default: กำหนดค่า default
- blank: ถ้ามีค่าเป็น True คือ column นี้มีค่าเป็น "" หรือ empty string ได้
- null: ถ้ามีค่าเป็น True คือ column นี้มีค่าเป็น null ได้
- unique: ถ้ามีค่าเป็น True คือ ค่าใน column นี้ห้ามซ้ำ

- primary_key: ถ้ามีค่าเป็น True คือ column นี้เป็น primary key ของ table (ถ้าไม่กำหนด Django จะสร้าง column ชื่อ `id` ให้อัตโนมัติเป็น primary key)

Field options

- db_index: ถ้ามีค่าเป็น True คือจะสร้าง index ใน database สำหรับ column นี้

Python Datetime Module

DateTime module มีทั้งหมด 5 class หลัก

1. `date` – มี attributes ได้แก่ year, month, และ day
2. `time` – มี attributes ได้แก่ hour, minute, second, microsecond, และ tzinfo
3. `datetime` – คือการรวม `date` และ `time` และมี attributes ได้แก่ year, month, day, hour, minute, second, microsecond, and tzinfo
4. `timedelta` – เป็นระยะเวลา (microsecond) ซึ่งเป็นส่วนต่างของ 2 date, time หรือ datetime
5. `tzinfo` – เป็น object สำหรับเก็บข้อมูล time zone

datetime

Argument `year`, `month`, และ `day` จะต้องกำหนด (mandatory) และ `tzinfo` เป็น None ได้

Range ของ attribute แต่ละตัว:

- MINYEAR <= year <= MAXYEAR
- 1 <= month <= 12
- 1 <= day <= number of days in the given month and year
- 0 <= hour < 24
- 0 <= minute < 60
- 0 <= second < 60
- 0 <= microsecond < 1000000

```
# datetime(year, month, day, hour, minute, second, microsecon
>>> a = datetime(1999, 12, 12, 12, 12, 12, 342380)
>>> print(a)
1999-12-12 12:12:12.342380
```

timedelta

เป็นคลาสในโมดูล `datetime` ของ Python ที่ใช้สำหรับการคำนวณเกี่ยวกับช่วงเวลา เช่น การบวกหรือการลบเวลาจากวันที่ใดๆ หรือการหาช่วงเวลาที่ต่างกันระหว่างสองวันที่

timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0, hours=0, weeks=0)

```
from datetime import datetime, timedelta

# กำหนดวันที่และเวลาเริ่มต้น
start_date = datetime(2023, 8, 19, 15, 0, 0)

# เพิ่ม 5 วันกับ 3 ชั่วโมงไปยังวันที่เริ่มต้น
new_date = start_date + timedelta(days=5, hours=3)

print(new_date)  # Output: 2023-08-24 18:00:00
```

```
>>> from datetime import datetime, timedelta

# Using current time
>>> ini_time_for_now = datetime.now()

# printing initial_date
>>> print("initial_date", str(ini_time_for_now))
initial_date 2024-07-13 23:45:16.572404

# Calculating future dates
# for two years
>>> future_date_after_2yrs = ini_time_for_now + timedelta(day

>>> future_date_after_2days = ini_time_for_now + timedelta(da

# printing calculated future_dates
>>> print('future_date_after_2yrs:', str(future_date_after_2y
future_date_after_2yrs: 2026-07-13 23:45:16.572404

>>> print('future_date_after_2days:', str(future_date_after_2
future_date_after_2days: 2024-07-15 23:45:16.572404
```

Python datetime.tzinfo()

The `datetime.now()` function จะไม่มีการเก็บข้อมูล time zones การเก็บข้อมูล time zines จะใช้ `tzinfo` ซึ่งเป็น abstract base case ใน Python

เราสามารถส่ง instance ของ class `tzinfo` ใน constructors ของ object datetime and time เพื่อใช้ในการกำหนด time zones

Naive and Aware datetime objects

- **Naive datetime objects** หมายถึง datetime object ที่ไม่มีการกำหนดข้อมูล time zone (tzinfo เป็น None)
- **Aware datetime objects** คือ datetime object ทีมีข้อมูล time zone

```
>>> from zoneinfo import ZoneInfo
>>> from datetime import datetime

>>> dt1 = datetime(2015, 5, 21, 12, 0)
>>> print(dt1)
2015-05-21 12:00:00
>>> dt2 = datetime(2015, 12, 21, 12, 0, tzinfo = ZoneInfo(key
>>> print(dt2)
2015-12-21 12:00:00+07:00

>>> print("Naive Object :", dt1.tzname())
Naive Object : None
>>> print("Aware Object :", dt2.tzname())
Aware Object : +07

>>> now_aware = dt1.replace(tzinfo=ZoneInfo(key='UTC'))
>>> print(now_aware)
2015-05-21 12:00:00+00:00
```

Django - Time zones

ถ้ามีการ enable time zone support (USE_TZ=True) โดย default Django จะบันทึก datetime information ในฐานข้อมูลเป็น UTC เมื่อ query ข้อมูลออกมาก็จะได้เป็น object datetime ที่ time-zone-aware

▼ Using Jupyter Notebook

การใช้งาน Django ใน Jupyter Notebook สามารถทำตามขั้นตอนดังนี้

1. ติดตั้ง virtualenv โดยกำหนด version ของภาษา python ดังนี้

   สำหรับ Windows

   ```
   py -m venv myvenv
   ```

   สำหรับ MAC OS

   ```
   python3 -m venv myvenv4
   ```

   activate และติดตั้ง Django และ psycopg2

   ```
   pip install django psycopg2-binary
   ```

2. สร้าง project Django
3. ติดตั้ง `django-extensions` และ `jupyter notebook` ด้วยคำสั่ง

   ```
   pip install django-extensions ipython jupyter notebook
   ```

4. จากนั้นให้แก้ไข version ของ package ภายใน jupyter และ notebook

   ```
   pip install ipython==8.25.0 jupyter_server==2.14.1 jupyt
   erlab==4.2.2 jupyterlab_server==2.27.2
   ```

   แก้ไข version notebook

   ```
   pip install notebook==6.5.6
   ```

   หากติดตั้ง หรือ run jupyter ไม่ได้ให้ลองเปลี่ยน notebook version ดังนี้ `6.5.7`

5. จากนั้นสร้าง directory ชื่อ `notebooks`

```
mkdir notebooks
```

6. เพิ่ม `django-extensions` ใน INSTALLED_APPS ในไฟล์ settings.py

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",

    "django_extensions",
    "blogs",
]
```

7. ทำการ start Jupyter Notebook server ด้วย command

```
python manage.py shell_plus --notebook
```

ซึ่งจะเปิด Jupyter Notebook ขึ้นมาใน Web Browser

1. เข้าไปที่ folder `notebooks`
2. สร้าง ไฟล์ ipynb สำหรับใช้กับ project django
3. จากนั้นใน Cell แรกของไฟล์ Notebook เพิ่ม code นี้ลงไป

```
import os
os.environ["DJANGO_ALLOW_ASYNC_UNSAFE"] = "true"
```

4. สามารถทำการ import models และ query ข้อมูลโดยใช้ API ของ Django ได้เลย

```
from blogs.models import Blog
```

```
for blog in Blog.objects.all():
    print(blog)
```

▼ Week 4

Topic: Making Queries & Field Lookups

การ `SELECT` ข้อมูลออกมาจาก database นั้นทาง Django มี API ให้เราใช้งานได้ง่ายและสามารถทำ query ที่ซับซ้อนได้ด้วย โดยเราจะได้ instance ของ class `Queryset` มาใช้งาน

```
# SELECT * FROM entry;
Entry.objects.all()

# ทีนี้เรามาลองเพิ่ม filter conditions กันบ้าง (ซึ่งก็คือการ SELECT โดยใส่เงื่อ
Entry.objects.filter(pub_date__year=2010)
```

- เราสามารถ chain method `filter()` และ `exclude()` ได้

  NOTE: `exclude()` คือการใส่เงื่อนไขที่จะกรองข้อมูลออก ดังในตัวอย่างด้านล่างคือการกรองข้อมูล blog entries หลังจากวันปัจจุบันออก

  ```
  Entry.objects.filter(headline__startswith="What").exclud
  e(
  ...     pub_date__gte=datetime.date.today()
  ... ).filter(pub_date__gte=datetime.date(2005, 1, 30))
  ```

In Query SQL

```
SELECT * FROM entry WHERE entry.headline LIKE "What%" AND ent
```

- การใช้ method `filter()` จะ return `QuerySet` ออกมาเสมอ แม้ว่า record ของข้อมูลที่ได้จากการ filter จะมีเพียง 1 record ก็จะได้ `QuerySet` ที่มีข้อมูล 1 แถวออกจาก ดังนั้นถ้าเรา

ต้องการที่จะได้ instance ของ class นั้นมาใช้งานเลย (ไม่ใช่ `QuerySet`) เราจะต้องใช้ `get()`

```
>>> one_entry = Entry.objects.get(pk=1)
>>> one_entry = Entry.objects.filter(pk=1).first()
>>> one_entry = Entry.objects.filter(pk=1)[0]
>>> # ทั้ง 3 บรรทัดนี้ให้ผลเหมือนกัน
```

- ในกรณีที่เราต้อง SELECT และต้องการ LIMIT ผลลัพธ์เราสามารถทำได้คล้ายๆ กับการ slice list ของ Python

  NOTE: การใช้ negative index (`Entry.objects.all()[-1]`) นั้นไม่ support

```
>>> Entry.objects.all()[:5] # LIMIT 5

>>> Entry.objects.all()[5:10] # OFFSET 5 LIMIT 5
```

- การเปรียบเทียบ instance ของ model 2 ตัวว่าเป็นตัวเดียวกันไหม สามารถทำได้โดยใช้ `==`

```
>>> some_entry == other_entry
>>> some_entry.id == other_entry.id
```

- การลบข้อมูลออกจาก database สามารถทำได้โดยใช้ method `delete()`
  - ลบทีละตัว

    ```
    >>> e.delete()
    (1, {'blog.Entry': 1})
    ```

  - ลบทีละหลายตัว

    ```
    >>> Entry.objects.filter(pub_date__year=2005).delete
    ()
    ```

```
(5, {'blog.Entry': 5})
```

- **Copying model instances**

  เมื่อเราตั้ง `instance.pk = None` และ `instance._state.adding = True` ก่อนจะเซฟ (บันทึก) ข้อมูล Django จะสร้าง object ใหม่ในฐานข้อมูลที่มีข้อมูลเหมือนของเดิมเป๊ะๆ แต่เป็นรายการใหม่ที่มีเลข id ใหม่

```
blog = Blog(name="My blog", tagline="Blogging is easy")
blog.save()  # blog.pk == 1

blog.pk = None
blog._state.adding = True
blog.save()  # blog.pk == 2
```

- เราสามารถทำได้โดยใช้ `raw()` method เมื่อต้องการเขียน SQL Query เอง
- ผลลัพธ์ที่ได้จาก `raw()` เป็น instance ของ class `django.db.models.query.RawQuerySet` ซึ่งใช้งานคล้ายกับ `QuerySet` ที่เราคุ้นเคย

```
# สมมติเรามี model Person
class Person(models.Model):
    first_name = models.CharField(...)
    last_name = models.CharField(...)
    birth_date = models.DateField(...)

# สามารถเขียน SELECT query ได้ดังนี้
>>> for p in Person.objects.raw("SELECT * FROM myapp_person")
...     print(p)
...
John Smith
Jane Jones

# สมมติชื่อ field ไม่ตรงกับที่ประกาศใน model
```

```
>>> name_map = {"first": "first_name", "last": "last_name", "
>>> Person.objects.raw("SELECT * FROM some_other_table", tran
```

Field Lookups

Django Doc

ความสุดยอดของ Django คือการที่เราสามารถเขียน WHERE condition ในการ query ข้อมูลจาก database ได้อย่างง่ายดายโดยการกำหนด lookup type ใน methods `filter()`, `exclude()` และ `get()`

โดย format การกำหนด lookup type เป็นดังนี้ `field__lookuptype=value` (underscore 2 ตัว ระหว่าง ชื่อ field และ lookup type)

ในกรณีที่ต้องการ filter ด้วย foreign key จะต้องเติม `_id` ต่อท้ายชื่อ field ด้วย เช่น

```
>>> Entry.objects.filter(blog_id=4)
```

Field lookup reference

- exact
- iexact
- contains

```
-- Entry.objects.filter(headline__contains='Lennon')
SELECT ... WHERE headline LIKE '%Lennon%';
```

- icontains

```
-- Entry.objects.filter(headline__icontains='Lennon')
SELECT ... WHERE headline ILIKE '%Lennon%';
```

- startswith
- endswith

- in

```
-- Entry.objects.filter(headline__in=('a', 'b', 'c'))
SELECT ... WHERE headline IN ('a', 'b', 'c');
```

- gt, gte, lt, lte

```
SELECT ... WHERE id > 4;
```

- range

```
-- Entry.objects.filter(pub_date__range=(start_date, end_da
te))
SELECT ... WHERE pub_date BETWEEN '2005-01-01' AND '2005-03
-31';
```

- date, year, month, day, week, week_day

```
-- Entry.objects.filter(pub_date__year=2005)
-- Entry.objects.filter(pub_date__year__gte=2005)
SELECT ... WHERE pub_date BETWEEN '2005-01-01' AND '2005-12
-31';
SELECT ... WHERE pub_date >= '2005-01-01';
```

- isnull

```
-- Entry.objects.filter(pub_date__isnull=True)
SELECT ... WHERE pub_date IS NULL;
```

- regex

```
-- Entry.objects.get(title__regex=r"^(An?|The) +")
SELECT ... WHERE title ~ '^(An?|The) +'; -- PostgreSQL
```

HINT: ถ้าอยากลองพิมพ์ SQL query ออกมาดูสามารถทำได้โดยใช้ `.query`

```
q = Entry.objects.filter(headline__in=('a', 'b', 'c'))

print(q.query)
```

Lookups that span relationships

QuerySet API ของ Django ช่วยให้เราสามารถ query ข้อมูลที่เกี่ยวข้องกับตารางอื่นที่มี relationship กันได้อย่างสะดวก โดย Django จะไปจัดการเรื่องการ generate SQL JOINs ให้หลังบ้าน

ยกตัวอย่างเช่น ถ้าเราต้องการดึงข้อมูล Entry ทั้งหมดของ Blog ที่มี name = "Beatles Blog"

```
>>> Entry.objects.filter(blog__name="Beatles Blog")
>>> Entry.objects.filter(blog__name__contains="Beatles Blo
g")
```

สังเกตว่าเราเพียงเอาชื่อ field foreign key มาต่อด้วยชื่อ field ของตารางที่อ้างอิงไปถึง โดยคั่นด้วย underscore 2 ตัว - `blog__name` - และยังสามารถต่อด้วย lookup type ได้อีก

นอกจากนั้นเรายังสามารถ query ข้อมูลไปกี่ต่อก็ได้ ดังตัวอย่าง

```
Blog.objects.filter(entry__authors__name="Lennon")
Blog.objects.filter(entry__authors__name__isnull=True)
```

Filters can reference fields on the model

ในกรณีที่เราต้องการเปรียบเทียบค่าของ field ใน model กับ field อื่นใน model เดียวกัน เราสามารถใช้ **F expressions** ได้ `F()`

```
>>> from django.db.models import F
>>> Entry.objects.filter(number_of_comments__gt=F("number_o
f_pingbacks"))
>>> Entry.objects.filter(authors__name=F("blog__name")) # s
pan relationships
```

โดย Django นั้น support การใช้ +, -, *, / ร่วมกับ `F()` ด้วย เช่น

```
>>> Entry.objects.filter(number_of_comments__gt=F("number_o
f_pingbacks") * 2)
>>> Entry.objects.filter(rating__lt=F("number_of_comments")
+ F("number_of_pingbacks"))
```

Complex lookups with Q objects

Keyword argument ที่ส่งเข้าไปใน method `filter()` ทุกตัวจะถูกเอามา generate เป็น `SELECT ... WHERE ... AND ...` เสมอ เช่น

โดยปกติถ้าเราใช้ `,` ขั้นระหว่าง filter condition จะเป็นการ AND กัน

```
-- Entry.objects.filter(headline__contains='Lennon', pub_da
te__year=2005)
SELECT * FROM entry WHERE headline LIKE '%Lennon%' AND pub_
date BETWEEN '2005-01-01' AND '2005-12-31';
```

ในกรณีที่เราต้องการทำการ query ที่ซับซ้อน อาจจะต้องการใช้ `OR` หรือ `NOT` ร่วมด้วย เราจะต้องใช้ `Q objects`

กรณี OR

```
Entry.objects.filter(Q(headline__startswith="Who") | Q(head
line__startswith="What"))
# SELECT ... WHERE headline LIKE 'Who%' OR headline LIKE 'W
hat%'
```

กรณี NOT

```
Entry.objects.filter(Q(headline__startswith="Who") | ~Q(pub
_date__year=2005))
# SELECT ... WHERE headline LIKE 'Who%' OR pub_date NOT BET
WEEN '2005-01-01' AND '2005-12-31';
```

กรณี nested conditions

```
Poll.objects.get(
    Q(question__startswith="Who"),
    (Q(pub_date=date(2005, 5, 2)) | Q(pub_date=date(2005,
5, 6))),
)
```

แปลงเป็น SQL

```
SELECT * from polls WHERE question LIKE 'Who%'
    AND (pub_date = '2005-05-02' OR pub_date = '2005-05-0
6')
```

▼ Week 5

Topic:

1. Query Expression
2. Many-to-one Relationships
3. One-to-one Relationships
4. Many-to-many Relationships

Query Expression

Django Doc

ORM ของ Django นั้น support การใช้งาน function ในการคำนวณ (+, -, *, /) การ aggregate ต่างๆ เช่น SUM(), MIN(), MAX(), COUNT(), AVG() และการทำ Subquery

ซึ่งในส่วนนี้เราจะเรียกว่าการทำ Query Expression

สมมติเรามี model `Company` ดังนี้

```
from datetime import date

from django.db import models
```

```
class Company(models.Model):
    name = models.CharField(max_length=100)
    ticker = models.CharField(max_length=20, null=True)
    num_employees = models.IntegerField()
    num_tables = models.IntegerField()
    num_chairs = models.IntegerField()

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

เรามา setup project กันสำหรับ tutorial นี้

1. สร้าง project ใหม่ชื่อ `week5_tutorial` (สร้าง vitual environment ใหม่ด้วย)
2. สร้าง app ชื่อ `companies` และทำการตั้งค่าใน `settings.py`
3. แก้ไขไฟล์ `/companies/models.py` และเพิ่ม code ด้านบนลงไป โดย models นี้เราจะใช้ในการทำ tutorial วันนี้กัน
4. `makemigrations` และ `migrate`

เปิด Django shell จากนั้นสร้างแถวข้อมูลด้วยคำสั่ง

```
>>> from company.models
>>> Company.objects.create(name="Company AAA", num_employee
s=120, num_chairs=150, num_tables=60)
>>> Company.objects.create(name="Company BBB", num_employee
s=50, num_chairs=30, num_tables=20)
>>> Company.objects.create(name="Company CCC", num_employee
s=100, num_chairs=40, num_tables=40)
```

ลองทดสอบคำสั่งด้านล่างนี้ดู แล้วดูสิว่าได้ผลลัพธ์เป็นอย่างไร

```
>>> from company.models import Company
>>> from django.db.models import Count, F, Value
>>> from django.db.models.functions import Length, Upper
>>> from django.db.models.lookups import GreaterThan

# Find companies that have more employees than chairs.
```

```
>>> Company.objects.filter(num_employees__gt=F("num_chair
s"))

# Find companies that have at least twice as many employees
as chairs.
>>> Company.objects.filter(num_employees__gt=F("num_chair
s") * 2)

# Find companies that have more employees than the number o
f chairs and tables combined.
>>> Company.objects.filter(num_employees__gt=F("num_chair
s") + F("num_tables"))

# How many chairs are needed for each company to seat all e
mployees?
>>> company = (
...     Company.objects.filter(num_employees__gt=F("num_cha
irs"))
...     .annotate(chairs_needed=F("num_employees") - F("num
_chairs"))
...     .first()
... )
>>> company.num_employees
50
>>> company.num_chairs
30
>>> company.chairs_needed
20

# Create a new company using expressions.
>>> company = Company.objects.create(name="Google", ticker=
Upper(Value("goog")))
# Be sure to refresh it if you need to access the field.
>>> company.refresh_from_db()
>>> company.ticker
'GOOG'
```

```
# Expressions can also be used in order_by(), either directly
>>> Company.objects.order_by(Length("name").asc())
>>> Company.objects.order_by(Length("name").desc())

# Lookup expressions can also be used directly in filters
>>> Company.objects.filter(GreaterThan(F("num_employees"), F("num_chairs")))
# or annotations.
>>> Company.objects.annotate(
...     need_chairs=GreaterThan(F("num_employees"), F("num_chairs")),
... )
```

Aggregate expression

สำหรับ tutorial นี้ให้ทำตามขั้นตอนนี้

1. สร้าง app ใหม่ชื่อ `books` ใน project `week5_tutorial` อันเดิม
2. เพิ่ม app books ใน `settings.py`
3. แก้ไขไฟล์ `/books/models.py` และเพิ่ม code ด้านล่างลงไป โดย models เหล่านี้เราจะใช้ในการทำ tutorial นี้กัน
4. `makemigrations` และ `migrate`

```
from django.db import models


class Author(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()


class Publisher(models.Model):
    name = models.CharField(max_length=300)
```

```
class Book(models.Model):
    name = models.CharField(max_length=300)
    pages = models.IntegerField()
    price = models.DecimalField(max_digits=10, decimal_plac
es=2)
    rating = models.FloatField()
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher, on_delete=mode
ls.CASCADE)
    pubdate = models.DateField()


class Store(models.Model):
    name = models.CharField(max_length=300)
    books = models.ManyToManyField(Book)
```

ทำการ import ข้อมูลเข้าตารางทั้งหมดด้วย SQL ในไฟล์ `books.sql`

ลองทดสอบคำสั่งด้านล่างนี้ดู แล้วดูสิว่าได้ผลลัพธ์เป็นอย่างไร

```
>>> from books.models import Book
# Total number of books.
>>> Book.objects.count()
59

# Total number of books with publisher=Penguin Books
>>> Book.objects.filter(publisher__name="Penguin Books").co
unt()
20

# Average price across all books, provide default to be ret
urned instead
# of None if no books exist.
>>> from django.db.models import Avg
```

```
>>> Book.objects.aggregate(Avg("price", default=0))
{'price__avg': Decimal('9.7018644067796610')}

# Max price across all books, provide default to be returne
d instead of
# None if no books exist.
>>> from django.db.models import Max
>>> Book.objects.aggregate(Max("price", default=0))
{'price__max': Decimal('14.99')}

# All the following queries involve traversing the Book<->P
ublisher
# foreign key relationship backwards.

# Each publisher, each with a count of books as a "num_book
s" attribute.
>>> from books.models import Publisher
>>> from django.db.models import Count
>>> pubs = Publisher.objects.annotate(num_books=Count("boo
k"))
>>> pubs
<QuerySet [<Publisher: BaloneyPress>, <Publisher: SalamiPre
ss>, ...]>
>>> pubs[0].num_books
20

# Each publisher, with a separate count of books with a rat
ing above and below 4
>>> from django.db.models import Q
>>> above = Publisher.objects.annotate(above_4=Count("boo
k", filter=Q(book__rating__gt=4)))
>>> below = Publisher.objects.annotate(below_4=Count("boo
k", filter=Q(book__rating__lte=4)))
>>> above[0].above_4
16
>>> below[0].below_4
```

```
4

# The top 5 publishers, in order by number of books.
>>> pubs = Publisher.objects.annotate(num_books=Count("boo
k")).order_by("-num_books")[:5]
>>> pubs[0].num_books
39
```

Model Relationships

- **Many-to-one relationships**

  สำหรับการนิยาม Many-to-one Relationships จะใช้การประกาศ ForeignKey เป็น field ใน models

  ต่อเนื่องจากตัวอย่าง `books/models.py` ใน model `Book` จะเห็นว่ามีการประกาศ ForeignKey เก็บ `publisher_id` ซึ่งชี้ไปยัง instance ใน model `Publisher`

  ```
  # /books/models.py
  ...
  class Publisher(models.Model):
      name = models.CharField(max_length=300)

  class Book(models.Model):
      name = models.CharField(max_length=300)
      pages = models.IntegerField()
      price = models.DecimalField(max_digits=10, decimal_p
  laces=2)
      rating = models.FloatField()
      authors = models.ManyToManyField(Author)
      publisher = models.ForeignKey(Publisher, on_delete=m
  odels.CASCADE)
      pubdate = models.DateField()
  ...
  ```

  สมมติเราจะสร้าง book เล่มใหม่ที่มีการ FK ไปหา publisher "Penguin Books"

```
>>> from books.models import Publisher, Book
>>> from datetime import datetime
# Get the publisher instance
>>> penguin_pub = Publisher.objects.get(name="Penguin Bo
oks")

# Create a new book
>>> book = Book.objects.create(
    name="Web Programming is HARD",
    pages=200,
    price=10.00,
    rating=4.5,
    publisher=penguin_pub,
    pubdate=datetime.now().date()
)
>>> book
<Book: Book object (61)>
>>> book.publisher.name
'Penguin Books'
>>> book.publisher.id
1
```

ในกรณีที่เราต้องการรายการ book ทั้งหมดที่เกี่ยวข้องกับ publisher "Penguin Books" สามารถทำได้ดังนี้

```
>>> from books.models import Publisher, Book
# Get the publisher instance
>>> penguin_pub = Publisher.objects.get(name="Penguin Bo
oks")

# Get all books published by "Penguin Books"
>>> books = penguin_pub.book_set.all()
<QuerySet [<Book: Book object (2)>, <Book: Book object
(3)>, <Book: Book object (4)>, <Book: Book object (5)>,
<Book: Book object (6)>, <Book: Book object (7)>, <Book:
```

```
Book object (8)>, <Book: Book object (9)>, <Book: Book o
bject (10)>, <Book: Book object (11)>, <Book: Book objec
t (12)>, <Book: Book object (13)>, <Book: Book object (1
4)>, <Book: Book object (15)>, <Book: Book object (16)>,
<Book: Book object (17)>, <Book: Book object (18)>, <Boo
k: Book object (19)>, <Book: Book object (20)>, <Book: B
ook object (21)>, '...(remaining elements truncate
d)...']>

# How may books?
>>> penguin_pub.book_set.count()
21

# Get top 10 best rating books
>>> penguin_pub.book_set.order_by("-rating")[:10]
<QuerySet [<Book: Book object (14)>, <Book: Book object
(4)>, <Book: Book object (15)>, <Book: Book object (9)>,
<Book: Book object (12)>, <Book: Book object (3)>, <Boo
k: Book object (8)>, <Book: Book object (18)>, <Book: Bo
ok object (61)>, <Book: Book object (10)>]>

# Get books with name starting with "The"
>>> penguin_pub.book_set.filter(name__startswith="The")
<QuerySet [<Book: Book object (2)>, <Book: Book object
(6)>, <Book: Book object (9)>, <Book: Book object (15)>,
<Book: Book object (18)>]>

# Get only ids
>>> penguin_pub.book_set.filter(name__startswith="The").
values_list("id", flat=True)
<QuerySet [2, 6, 9, 15, 18]>

# Get id and name
>>> penguin_pub.book_set.filter(name__startswith="The").
values("id", "name")
<QuerySet [{'id': 2, 'name': 'The Great Gatsby'}, {'id':
```

```
6, 'name': 'The Catcher in the Rye'}, {'id': 9, 'name':
'The Odyssey'}, {'id': 15, 'name': 'The Hobbit'}, {'id':
18, 'name': 'The Hitchhiker Guide to the Galaxy'}]>
```

สมมติว่าเราต้องการต้นหาจากทางฝั่ง book บ้าง ถ้าเราต้องหนังสือที่ rating >= 4.5 และ published โดยสำนักพิมพ์ "Oxford University Press"

```
>>> from books.models import Book

>>> results = Book.objects.filter(publisher__name="Oxfor
d University Press", rating__gte=4.5)
>>> results
<QuerySet [<Book: Book object (24)>, <Book: Book object
(27)>, <Book: Book object (30)>, <Book: Book object (33)
>, <Book: Book object (44)>, <Book: Book object (56)>]>
```

หรือจะ filter จากทางฝั่ง publisher ก็ได้

```
>>> from books.models import Publisher
>>> Publisher.objects.filter(book__id=20)
<QuerySet [<Publisher: Publisher object (1)>]>

>>> Publisher.objects.filter(book__pubdate='1967-05-30')
<QuerySet [<Publisher: Publisher object (1)>]>
# SELECT "books_publisher"."id", "books_publisher"."nam
e" FROM "books_publisher" INNER JOIN "books_book" ON ("b
ooks_publisher"."id" = "books_book"."publisher_id") WHER
E "books_book"."pubdate" = 1967-05-30
```

- **One-to-one Relationships**

  เรามาเพิ่มความสัมพันธ์แบบ one-to-one ให้กับตารางใน database ของเรากัน

  แก้ไขเพิ่ม code ด้านล่างนี้ในไฟล์ `/books/models.py`

```
...
# เพิ่มไว้ล่างสุดเลยนะครับ
class StoreContact(models.Model):
    mobile = models.CharField(max_length=20)
    email = models.EmailField(max_length=50, blank=True,
null=True)
    address = models.TextField()
    store = models.OneToOneField(Store, on_delete=model
s.CASCADE)
```

เรามาสร้างเพิ่ม store contact กันสำหรับ "KMITL Book Store"

```
>>> from books.models import Store, StoreContact
# Get KMITL Book Store
>>> store = Store.objects.get(name="KMITL Book Store")
# Create contact information
>>> contact = StoreContact(
    mobile="021113333",
    email="book_shop@it.kmitl.ac.th",
    address="KMITL",
    store=store
)
>>> contact.save()
```

การเข้าถึงข้อมูล สามารถทำได้จากทั้ง 2 ฝั่ง

```
>>> contact.store
<Store: Store object (2)>

>>> store.storecontact
<StoreContact: StoreContact object (1)>

# สามารถ filter ได้คล้ายกับ one-to-many
>>> Store.objects.filter(storecontact__mobile="02111333
```

```
3")
<QuerySet [<Store: Store object (2)>]>
```

- **Many-to-many Relationships**

  จะเห็นได้ว่า model Book และ Author มีความสัมพันธ์กันแบบ many-to-many

```
class Author(models.Model):
    name = models.CharField(max_length=100)
    age = models.IntegerField()

...

class Book(models.Model):
    name = models.CharField(max_length=300)
    pages = models.IntegerField()
    price = models.DecimalField(max_digits=10, decimal_p
laces=2)
    rating = models.FloatField()
    authors = models.ManyToManyField(Author)
    publisher = models.ForeignKey(Publisher, on_delete=m
odels.CASCADE)
    pubdate = models.DateField()
```

  เมื่อเราสั่ง `python manage.py migrate` Django จะทำการสร้างตารางกลางให้อัตโนมัติ อย่างในกรณีนี้จะมีตาราง `books_book_author` ถูกสร้างขึ้นมา

  สำหรับการเพิ่ม author ให้กับ book

```
>>> from books.models import Book, Author
>>> a1 = Author.objects.get(pk=1)
>>> a2 = Author.objects.get(pk=2)

>>> book = Book.objects.get(pk=10)
>>> book.authors.add(a1, a2)
```

```
>>> book.authors.all()
<QuerySet [<Author: Author object (5)>, <Author: Author
object (1)>, <Author: Author object (2)>]>
```

สำหรับการเพิ่ม book ใหม่ให้กับ author

```
>>> from books.models import Book, Author
>>> b1 = Book.objects.get(pk=11)
>>> b2 = Book.objects.get(pk=12)

>>> author = Author.objects.get(pk=10)
>>> author.book_set.add(b1, b2)

>>> author.book_set.all()
<QuerySet [<Book: Book object (11)>, <Book: Book object
(12)>, <Book: Book object (19)>, <Book: Book object (20)
>, <Book: Book object (30)>, <Book: Book object (50)>]>
```

สามารถทำการ filter ได้จากทั้ง 2 ฝั่งเช่นเดียวกับ one-to-one และ one-to-many

```
>>> Book.objects.filter(authors__name="F. Scott Fitzgera
ld")
<QuerySet [<Book: Book object (51)>, <Book: Book object
(2)>, <Book: Book object (21)>, <Book: Book object (31)
>, <Book: Book object (10)>]>

>>> Author.objects.filter(book__name="Crime and Punishme
nt")
<QuerySet [<Author: Author object (5)>, <Author: Author
object (1)>, <Author: Author object (2)>]>
```

สามารถทำการ ยกเลิก ความสัมพันธ์ ได้โดยใช้ `remove()` หรือ `clear()` ถ้าต้องการลบความสัมพันธ์ทั้งหมด

```
>>> book = Book.objects.get(pk=10)

>>> book.authors.remove(a1)
>>> book.authors.all()
<QuerySet [<Author: Author object (2)>, <Author: Author
object (5)>]>

>>> book.authors.clear()
>>> book.authors.all()
<QuerySet []>
```

▼ Week 7

Topic: Views เน้นๆ

- **How Django processes a request**

```
from django.urls import path

from . import views

urlpatterns = [
    path("articles/2003/", views.special_case_2003),
    path("articles/<int:year>/", views.year_archive),
    path("articles/<int:year>/<int:month>/", views.month
_archive),
    path("articles/<int:year>/<int:month>/<slug:slug>/",
views.article_detail),
]
```

ตัวอย่างการทำงาน

- ถ้ามี request มาที่ path `/articles/2005/03/` จะ match path ที่ 3 ซึ่งเรียก "views.month_archive"

- ถ้ามี request มาที่ path `/articles/2003/` จะ match path ที่ 1 ซึ่งเรียก "views.special_case_2003"
- แต่ว่า `/articles/2003` จะไม่ match เลยเพราะไม่มี '/'
- ส่วน `/articles/2003/03/building-a-django-site/` จะ match path สุดท้าย ซึ่งเรียก "views.article_detail"

Note: ไม่จำเป็นจะต้องมี / ตอนต้นของ path

- **Django มี path converter ให้ใช้งานดังนี้:**
  - **str** - Matches any non-empty string, excluding the path separator, '/'. - ซึ่งจะเป็นค่า default ถ้าไม่มีการกำหนด
  - **int** - Matches zero or any positive integer. Returns an int.
  - **slug** - Matches any slug string consisting of ASCII letters or numbers (สามารถใช้ '-' และ '_') เช่น building-your-1st-django-site.
  - **uuid** - Matches a formatted UUID (จะต้องมี '-' และตัวอักษรเป็นตัวเล็กทุกตัว) เช่น 075194d3-6885-417e-a8a8-6c931e272f00 โดยค่าที่ได้รับใน view จะเป็น instance ของ UUID
  - **path** - Matches any non-empty string, including the path separator, '/'.

- **Using regular expressions**

  ถ้าต้องการใช้ regex ในการกำหนด URL สามารทำได้โดยใช้ `re_path()` แทน `path()`

  โดย format จะเป็น "(?Ppattern)" โดยที่ `name` คือเป็นชื่อตัวแปร และ `pattern` เป็น pattern ที่ต้องการ match

```
from django.urls import path, re_path

from . import views

urlpatterns = [
    path("articles/2003/", views.special_case_2003),
    re_path(r"^articles/(?P<year>[0-9]{4})/$", views.yea
```

```
r_archive),
    re_path(r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-
9]{2})/$", views.month_archive),
    re_path(
        r"^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]
{2})/(?P<slug>[\w-]+)/$",
        views.article_detail,
    ),
]
```

- **Including other URLconfs**

  โดยปกติเราจะแบ่งไฟล์ urls.py ไปอยู่แต่ละ app เพื่อความเป็นสัดส่วน ดังนั้นเราสามารถใช้ `include()` ในการบอก Django ได้ว่าให้ไป match URL ที่ไฟล์ไหน

```
from django.urls import include, path

urlpatterns = [
    # ... snip ...
    path("shop/", include("shop.urls")),
    path("contact/", include("contact.urls")),
    # ... snip ...
]
```

Writing Views

- **A simple view - function-based views**

```
from django.http import HttpResponse
import datetime


def current_datetime(request):
```

```
    now = datetime.datetime.now()
    html = "<html><body>It is now %s.</body></html>" % n
ow
    return HttpResponse(html)
```

Important! อย่าลืมว่าเราจะเรียก view นี้ได้จะต้องไป map URL มาที่ view นี้ก่อนนะครับ ใน urls.py

- **Returning errors**

  โดยปกติเมื่อเรา return HttpResponse() Django จะ default HTTP status code = 200 ให้ แต่ถ้าเราต้องการกำหนด status code เองก็สามารถทำได้

  ```
  from django.http import HttpResponse


  def my_view(request):
      # ...

      # Return a "created" (201) response code.
      return HttpResponse(status=201)
  ```

- **The Http404 exception**

  เพื่อความง่าย Django มีหน้า page 404 standard มาให้ (แต่เราสามารถ custom เองได้นะ) โดยถ้าคุณมีการ raise Http404 ตรงไหนใน view ก็ตาม Django จะ catch error นี้ให้และ return page 404 ตัว standard มาให้เสมอ พร้อม HTTP status code = 404

  ```
  from django.http import Http404
  from django.shortcuts import render
  from polls.models import Poll


  def detail(request, poll_id):
  ```

```
    try:
        p = Poll.objects.get(pk=poll_id)
    except Poll.DoesNotExist:
        raise Http404("Poll does not exist")
    return render(request, "polls/detail.html", {"poll"
```

View decorators

What is a decorator?

Source

ก่อนที่จะเข้าใจ decorator คุณจะต้องเข้าใจ concept เหล่านี้ก่อน

1. First-Class Objects - This means that functions can be passed around and used as arguments, just like any other object like str, int, float, list, and so on.

```
def say_hello(name):
    return f"Hello {name}"

def be_awesome(name):
    return f"Yo {name}, together we're the awesomes
t!"

def greet_bob(greeter_func):
    return greeter_func("Bob")
```

เมื่อเรียกใช้งาน

```
>>> greet_bob(say_hello)
'Hello Bob'

>>> greet_bob(be_awesome)
'Yo Bob, together we're the awesomest!'
```

1. Inner Functions - It’s possible to define functions inside other functions. Such functions are called inner functions.

```
def parent():
    print("Printing from parent()")

    def first_child():
        print("Printing from first_child()")

    def second_child():
        print("Printing from second_child()")

    second_child()
    first_child()
```

เมื่อเรียกใช้งาน

```
>>> parent()
Printing from parent()
Printing from second_child()
Printing from first_child()
```

1. Functions as Return Values

```
def parent(num):
    def first_child():
        return "Hi, I'm Elias"

    def second_child():
        return "Call me Ester"

    if num == 1:
        return first_child
    else:
        return second_child
```

เมื่อเรียกใช้งาน

```
>>> first = parent(1)
>>> second = parent(2)

>>> first
<function parent.<locals>.first_child at 0x7f599f1e2e
18>

>>> second
<function parent.<locals>.second_child at 0x7f599dad5
268>
```

ค่าที่ return ออกมาจะเป็น function ซึ่งสามารถ execute ได้

```
>>> first()
'Hi, I'm Elias'>>> second()
'Call me Ester'
```

Simple Decorators in Python

ทีนี้เรามาลองดู decorator ง่ายๆ กัน

```
def decorator(func):
    def wrapper():
        print("Something is happening before the func
tion is called.")
        func()
        print("Something is happening after the funct
ion is called.")
    return wrapper

def say_whee():
    print("Whee!")

say_whee = decorator(say_whee)
```

เมื่อเรียกใช้งาน

```
>>> from hello_decorator import say_whee

>>> say_whee()
Something is happening before the function is called.
Whee!
Something is happening after the function is called.
```

> Put simply, a decorator wraps a function, modifying its behavior.

ลองมาสร้าง decorator ที่ทำประโยชน์อะไรสักหน่อยจะได้เข้าใจมากขึ้น

```
from datetime import datetime

def not_during_the_night(func):
    def wrapper():
        if 7 <= datetime.now().hour < 22:
            func()
        else:
            pass  # Hush, the neighbors are asleep
    return wrapper

def say_whee():
    print("Whee!")

say_whee = not_during_the_night(say_whee)
```

ถ้าเราเรียก say_whee() หลังเวลา 22.00 ก็จะไม่มีการ print "Whee!" ออกมา

โดย Python เขาก็มี syntax ในการเรียกใช้ decorator ที่ง่ายละเห็นแล้วจำได้ว่าเป็น decorator ก็คือการใช้ `@` ดังในตัวอย่าง

```
from datetime import datetime

def not_during_the_night(func):
```

```
    def wrapper():
        if 7 <= datetime.now().hour < 22:
            func()
        else:
            pass  # Hush, the neighbors are asleep
    return wrapper

@not_during_the_nightdef say_whee():
    print("Whee!")
```

Allowed HTTP methods decorators

- require_http_methods(request_method_list) ใช้กำหนดว่า view นั้นๆ รับ request ที่มี HTTP Method อะไรได้บ้าง

```
from django.views.decorators.http import require_http
_methods


@require_http_methods(["GET", "POST"])
def my_view(request):
    # I can assume now that only GET or POST requests
make it this far
    # ...
    pass
```

- require_GET()
- require_POST()
- require_safe() - Decorator to require that a view only accepts the GET and HEAD methods.

Other decorators

- `gzip_page()`
- `no_append_slash()`
- `cache_control(**kwargs)`

- `never_cache(view_func)`

Writing your first view

ก่อนจะไปสร้าง view เรามาทำการ setup project "week7_tutorial" และสร้าง app "polls" กันก่อน
และทำการ copy models เหล่านี้ลงไปที่
`polls/models.py`

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

    def __str__(self):
        return self.question_text


class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=
models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

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

จากนั้นก็ `makemigrations` และ `migrate` และทำการ import `polls.sql` เข้าไปใน database

Let's start!

สำหรับ poll application เราสร้างจะ views ดังต่อไปนี้

- Question “index” page – แสดงรายการ questions ล่าสุด

- Question “detail” page – แสดง question text และตัวเลือก
- Vote action – สำหรับทำการ vote

เรามาเริ่มต้นด้วยการเพิ่ม code ด้านล่างนี้ใน `/polls/views.py`

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("This is the index page of po
lls app")

def detail(request, question_id):
    return HttpResponse("You're looking at question %
s." % question_id)


def vote(request, question_id):
    return HttpResponse("You're voting on question %
s." % question_id)
```

กำหนด path สำหรับเข้าถึง `urls.py` ของ app polls ใน `week7_tutorial/urls.py`

```
...

urlpatterns = [
    path("admin/", admin.site.urls),
    path("polls/", include("polls.urls")),
]
```

กำหนด path url สำหรับเข้าถึง views ด้านบน `/polls/urls.py`

```
from django.urls import path

from . import views

urlpatterns = [
```

```
    # ex: /polls/
    path("", views.index, name="index"),
    # ex: /polls/5/
    path("<int:question_id>/", views.detail, name="de
tail"),
    # ex: /polls/5/vote/
    path("<int:question_id>/vote/", views.vote, name
="vote"),
]
```

หมายเหตุ: int:question_id เป็นการประกาศ path parameter ซึ่งจะรับค่าตัวแปรที่ถูกส่งมาใน url

Write views that actually do something

เรามาปรับแก้ไข view index() ให้ทำการ query ข้อมูล question 5 รายการล่าสุด เรียงตาม pub_date แบบจากมากไปน้อย

```
from django.http import HttpResponse
from django.shortcuts import render

from .models import Question


def index(request):
    latest_question_list = Question.objects.order_by
("-pub_date")[:5]
    context = {"latest_question_list": latest_questio
n_list}
    return render(request, "index.html", context)

# Leave the rest of the views (detail, results, vote)
unchanged
```

ใน view index() เราได้ทำการ return list ของ questions ออกมา และส่งต่อข้อมูลไปยัง `/polls/index.html`

เอ้ะแต่ไฟล์ `/polls/index.html` มันอยู่ไหน ไม่เห็นมี @_@

เราจะต้องไปสร้างไฟล์ **template** ก่อน โดยสร้าง folder `/polls/templates` และสร้างไฟล์ `/polls/templates/index.html` และเพิ่ม code ด้านล่าง

```
<html>
    <head>
    </head>
    <body>
        <h1>Lastest questions</h1>
        {% if latest_question_list %}
            <ul>
            {% for question in latest_question_list
%}
                <li><a href="/polls/{{ question.id
}}/">{{ question.question_text }}</a></li>
            {% endfor %}
            </ul>
        {% else %}
            <p>No polls are available.</p>
        {% endif %}
    </body>
</html>
```

แก้ไขไฟล์ `mysite/settings.py` เพิ่ม code ดังนี้

```
import os
SETTINGS_PATH = os.path.dirname(os.path.dirname(__fil
e__))

TEMPLATE_DIRS = (
    os.path.join(SETTINGS_PATH, 'templates'),
)
```

เรามาลอง start server ดูว่าหน้า index สามารถใช้งานได้ไหม

เปิด browser และพิมพ์ url `http://127.0.0.1:8000/polls/`
จะเห็นว่ามีรายการ questions แสดงขึ้นมา 5 รายการ

เรามาทำให้ view อื่นๆ ใช้งานได้กัน

```
from django.shortcuts import render, redirect

from .models import Question, Choice

def index(request):
    latest_question_list = Question.objects.order_by
("-pub_date")[:5]
    context = {"latest_question_list": latest_questio
n_list}
    return render(request, "index.html", context)

def detail(request, question_id):
    question = Question.objects.get(pk=question_id)
    return render(request, "detail.html", {
        "question": question,
        "choices": question.choice_set.all().order_by
("choice_text")
    })

def vote(request, question_id):
    question = Question.objects.get(pk=question_id)

    if request.method == "GET":
        return render(request, "vote.html", {
            "question": question,
            "choices": question.choice_set.all().orde
r_by("choice_text")
        })
    elif request.method == "POST":
        choice_id = request.POST.get('choice')
        choice = Choice.objects.get(pk=choice_id)
```

```
        choice.votes += 1
        choice.save()
        return redirect("detail", question_id=questio
n_id)
```

สร้างไฟล์ `/polls/templates/detail.html` และเพิ่ม code ด้านล่าง

```
<html>
    <head>
    </head>
    <body>
        <h1>Question: {{ question.question_text }}</h
1>
        <p>Publist date: {{ question.pub_date }}</p>
        <ul>
        {% for choice in choices %}
            <li>{{ choice.choice_text }} (Votes: {{ch
oice.votes}})</li>
        {% endfor %}
        </ul>
        <br/>
            <a href="{% url 'vote' question.id %}">
            <button>Let's Vote</button>
        </a>
    </body>
</html>
```

สร้างไฟล์ `/polls/templates/vote.html` และเพิ่ม code ด้านล่าง

```
<html>
    <head>
    </head>
    <body>
        <h1>Question: {{ question.question_text }}</h
1>
        <p>Publist date: {{ question.pub_date }}</p>
```

```
        <form action="/polls/{{ question.id }}/vote/"
method="POST">
            {%csrf_token%}
            <ul>
            {% for choice in choices %}
                <li>
                    <input type="radio" id="choice{{
forloop.counter }}" name="choice" value="{{ choice.id
}}">
                    <label for="choice{{ forloop.coun
ter }}">{{ choice.choice_text }}</label>
                </li>
            {% endfor %}
            </ul>
            <input type="submit" value ="VOTE" name
="submit"/>
        </form>
    </body>
</html>
```

Let's try class-based views

เรามาลองเปลี่ยนเป็น class-based views กันบ้างนะครับ จะเห็นว่า code เป็นระเบียบขึ้น (ผมคิดว่านะ...)

แก้ไขใน `polls.views`

```
from django.shortcuts import render, redirect
from django.views import View

from .models import Question, Choice


class IndexView(View):

    def get(self, request):
```

```
        latest_question_list = Question.objects.order
_by("-pub_date")[:5]
        context = {"latest_question_list": latest_que
stion_list}
        return render(request, "index.html", context)

class PollView(View):

    def get(self, request, question_id):
        question = Question.objects.get(pk=question_i
d)
        return render(request, "detail.html", {
            "question": question,
            "choices": question.choice_set.all()
        })

class VoteView(View):

    def get(self, request, question_id):
        question = Question.objects.get(pk=question_i
d)
        return render(request, "vote.html", {
            "question": question,
            "choices": question.choice_set.all()
        })

    def post(self, request, question_id):
        choice_id = request.POST.get('choice')
        choice = Choice.objects.get(pk=choice_id)
        choice.votes += 1
        choice.save()
        return redirect("detail", question_id=questio
n_id)
```

อย่าลืมไปแก้ไข `polls/urls.py` ด้วยนะครับเพิ่ม `.as_view()`

```
from django.urls import path

from . import views

urlpatterns = [
    # ex: /polls/
    path("", views.IndexView.as_view(), name="inde
x"),
    # ex: /polls/5/
    path("<int:question_id>/", views.PollView.as_view
(), name="detail"),
    # ex: /polls/5/vote/
    path("<int:question_id>/vote/", views.VoteView.as
_view(), name="vote"),
]
```

Class-based views

Class-based view นั้นโดยหลักแล้วจะมีไว้เพื่อให้เราสามารถกำหนด response สำหรับ HTTP request method ที่แตกต่างกันได้ แทนที่จะต้องมาใช้ if ดังเช่นในกรณี function-based view

```
from django.http import HttpResponse


def my_view(request):
    if request.method == "GET":
        # <view logic>
        return HttpResponse("result")
    elif request.method == "POST":
        # <view logic>
        return HttpResponse(status=201)
```

แต่ถ้าเป็น class-based view จะเขียนได้ดังนี้

```
from django.http import HttpResponse
from django.views import View


class MyView(View):
    def get(self, request):
        # <view logic>
        return HttpResponse("result")

    def post(self, request):
        # <view logic>
        return HttpResponse(status=201)
```

ในกรณีที่ใช้ class-based view ในไฟล์ urls.py จะต้องปรับนิดหน่อย โดยจะต้องเรียก `as_view()`

```
# urls.py
from django.urls import path
from myapp.views import MyView

urlpatterns = [
    path("about/", MyView.as_view()),
]
```

Handling forms with class-based views

การใช้งาน form ใน function-based view จะเป็นประมาณนี้

```
from django.http import HttpResponseRedirect
from django.shortcuts import render

from .forms import MyForm


def myview(request):
```

```
    if request.method == "POST":
        form = MyForm(request.POST)
        if form.is_valid():
            # <process form cleaned data>
            return HttpResponseRedirect("/success/")
    else:
        form = MyForm(initial={"key": "value"})

    return render(request, "form_template.html", {"fo
rm": form})
```

ส่วนถ้าใช้ใน class-based view จะเป็นดังนี้

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from django.views import View

from .forms import MyForm


class MyFormView(View):
    form_class = MyForm
    initial = {"key": "value"}
    template_name = "form_template.html"

    def get(self, request, *args, **kwargs):
        form = self.form_class(initial=self.initial)
        return render(request, self.template_name,
{"form": form})

    def post(self, request, *args, **kwargs):
        form = self.form_class(request.POST)
        if form.is_valid():
            # <process form cleaned data>
            return HttpResponseRedirect("/success/")
```

```
        return render(request, self.template_name,
{"form": form})
```

จะเห็นได้ว่ามีการแยกที่เป็นสัดส่วนระหว่าง GET และ POST และสามารถกำหนด class attributes ได้ด้วย

Decorating class-based views

```
from django.contrib.auth.decorators import login_requ
ired
from django.utils.decorators import method_decorator
from django.views.generic import TemplateView


class ProtectedView(TemplateView):
    template_name = "secret.html"

    @method_decorator(login_required)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)
```

หรือ

```
from django.views.decorators.cache import never_cache
from django.contrib.auth.decorators import login_requ
ired
from django.utils.decorators import method_decorator
from django.views.generic import TemplateView


@method_decorator(never_cache, name="dispatch")
@method_decorator(login_required, name="dispatch")
class ProtectedView(TemplateView):
    template_name = "secret.html"
```

▼ Week 8

Topic: Templates

▼ The Django template language

Templates

> A template is a text file. It can generate any text-based format (HTML, XML, CSV, etc.).

ด้านล่างเป็นตัวอย่าง template ที่มีการใช้งานทุกๆ concept ที่เราจะพูดถึงในวันนี้

```
{% extends "base_generic.html" %}

{% block title %}{{ section.title }}{% endblock %}

{% block content %}
<h1>{{ section.title }}</h1>

{% for story in story_list %}
<h2>
  <a href="{{ story.get_absolute_url }}">
    {{ story.headline|upper }}
  </a>
</h2>
<p>{{ story.tease|truncatewords:"100" }}</p>
{% endfor %}
{% endblock %}
```

Variables

การแสดงค่าตัวแปรใน tamplate จะใช้ {{ variable }}

โดยถ้าเราพยายามแสดงค่าตัวแปรที่ไม่มีการส่งมาใน template ค่า '' จะถูกแสดงแทน

Filters

Filters ใช้ในการ modify ค่าของตัวแปร เช่น {{ name|lower }} โดย `|` จะเป็น syntax การ apply filter เข้าไปกับตัวแปร

เราสามารถ chain filter หลายๆตัวได้ เช่น {{ text|escape|linebreaks }}

นอกจากนั้น filter บางตัวก็มีการรับ argument เช่น {{ bio|truncatewords:30 }}

built-in filter reference

Filter ที่ใช้บ่อยๆ:

```
{{ value|default:"nothing" }}

{{ value|length }}

{{ value|filesizeformat }}
<!--If value is 123456789, the output would be 117.7 MB.
-->
```

Tags

Syntax ของ tag จะเป็น {% %} โดย tag จะเกี่ยวข้องกับ control flow และ logic

และบาง tag จะต้องมีการเปิด และ ปิด tag

built-in tag reference

เช่น

for

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

if elif else

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

Comments

```
{# greeting #}hello

{# {% if foo %}bar{% else %} #}
```

Template inheritance

Template inheritance นั้นช่วยให้เราสามารถวางโครงสร้างของหน้าเพจ โดยการสร้าง base "skeleton" เช่น

ลองมาดูตัวอย่างการทำ template inheritance กัน

```
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css">
    <title>{% block title %}My amazing site{% endblock
%}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
```

```
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
        {% endblock %}
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

ไฟล์ `base.html` นี้ทำการกำหนดโครงสร้างของหน้าเวบเพจเป็น 3 ส่วนคือ

1. title
2. sidebar
3. content

โดยการใช้ tag **block**

หน้าเพจที่มา extend `base.html` นี้ไปอาจจะเป็นดังตัวอย่าง

```
{% extends "base.html" %}

{% block title %}My amazing blog{% endblock %}

{% block content %}
{% for entry in blog_entries %}
    <h2>{{ entry.title }}</h2>
    <p>{{ entry.body }}</p>
{% endfor %}
{% endblock %}
```

จะเห็นได้ว่ามีการใช้ tag **extends** - `{% extends "base.html" %}`

หน้า "My amazing blog" นี้จะถูก Django render เป็น HTML ออกมาดังนี้

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <link rel="stylesheet" href="style.css">
    <title>My amazing blog</title>
</head>

<body>
    <div id="sidebar">
        <ul>
            <li><a href="/">Home</a></li>
            <li><a href="/blog/">Blog</a></li>
        </ul>
    </div>

    <div id="content">
        <h2>Entry one</h2>
        <p>This is my first entry.</p>

        <h2>Entry two</h2>
        <p>This is my second entry.</p>
    </div>
</body>
</html>
```

จะเห็นว่าตัวลูกที่ extend `base.html` ไม่ได้ทำการกำหนด block "sidebar" ดังนั้น code ใน block นั้นจะเป็นตามค่าที่กำหนดใน `base.html`

Automatic HTML escaping

เมื่อเราทำการ render HTML จากข้อมูลที่ส่งมาจาก database มันมักจะมีความเสี่ยงที่ข้อมูลจะทำให้หน้าเพจ HTML เกิดปัญหา เช่น

```
Hello, {{ name }}
```

```
--------------

<!-- ถ้าค่าของตัวแปร name คือ -->
<script>alert('hello')</script>

-------------

<!-- เราก็จะ render ได้ -->
Hello, <script>alert('hello')</script>
```

หรือค่าของ name = <b>username ก็จะทำให้หน้า page ทั้งหมดต่อจากตรงนี้เป็น bold ทันที

ดังนั้นโดย default Django จะทำการ escape ค่าของตัวแปรที่เอามา render ใน templage ให้อัตโนมัติ

- `<` is converted to `<`
- `>` is converted to `>`
- `'` (single quote) is converted to `&#x27;`
- `"` (double quote) is converted to `&quot;`
- `&` is converted to `&`

How to turn it off

เราสามารถปิด auto-escape ได้หลายระดับ ทั้ง per-site, per-template level หรือ per-variable level

Variable level

```
This will be escaped: {{ data }}
This will not be escaped: {{ data|safe }}

-------

This will be escaped: &lt;b&gt;
This will not be escaped: <b>
```

Template block level

```
Auto-escaping is on by default. Hello {{ name }}

{% autoescape off %}
    This will not be auto-escaped: {{ data }}.

    Nor this: {{ other_data }}
    {% autoescape on %}
        Auto-escaping applies again: {{ name }}
    {% endautoescape %}
{% endautoescape %}
```

Accessing method calls

เราสามารถเรียก function ของ instance ที่เราส่งไป render ที่ template ได้ เช่น

```
{% for comment in task.comment_set.all %}
    {{ comment }}
{% endfor %}

-------

{{ task.comment_set.all.count }}
```

หรือถ้าเรามีการ define method ของ class เอาไว้ เราก็สามารถเรียก method ของ instance ของ class นั้นได้ใน template

```
<!-- in models.py -->
class Task(models.Model):
    def foo(self):
        return "bar"

-------
```

```
<!-- in template.html -->
{{ task.foo }}
```

หมายเหตุ: แต่เราจะไม่สามารถส่ง argument เข้า function ได้นะครับใน template

Custom tag and filter libraries

เราสามารถสร้าง customer tag และ filter ได้เอง ละก็มีคนอื่นที่เขาทำ libraries ของ third-party custom tags and filters มาให้ใช้งานด้วยเช่น `humanize`

วิธีติดตั้ง `humanize`:

1. pip install humanize
2. เพิ่ม 'django.contrib.humanize' ใน INSTALLED_APPS ในไฟล์ settings

เช่น

```
{% load humanize %}

{{ 45000|intcomma }}
```

▼ Template tags ออกข้อสอบนิดหน่อย

Built-in template tags and filters

Built-in tag reference

Ref

- **autoescape** สำหรับเอาไว้ เปิด - ปิด auto escape

```
{% autoescape on %}
    {{ body }}
{% endautoescape %}
```

- **block**
- **comment**

```
<p>Rendered text with {{ pub_date|date:"c" }}</p>
{% comment "Optional note" %}
    <p>Commented out text with {{ create_date|date:"c"
}}</p>
{% endcomment %}
```

- **csrf_token**
- **cycle**

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

- **extends**
- **for**

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>

<!-- if "data" is a dictionary -->
{% for key, value in data.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

Variable	Description
`forloop.counter`	The current iteration of the loop (1-indexed)
`forloop.counter0`	The current iteration of the loop (0-indexed)

`forloop.revcounter`	The number of iterations from the end of the loop (1-indexed)
`forloop.revcounter0`	The number of iterations from the end of the loop (0-indexed)
`forloop.first`	True if this is the first time through the loop
`forloop.last`	True if this is the last time through the loop
`forloop.parentloop`	For nested loops, this is the loop surrounding the current one

ตัวอย่างการใช้งาน

```
<ul>
{% for athlete in athlete_list %}
    <li>{{forloop.counter}}. {{ athlete.name }}</li>
{% endfor %}
</ul>
```

- **for ... empty**

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>Sorry, no athletes in this list.</li>
{% endfor %}
</ul>
```

- **if**

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
```

```
    No athletes.
{% endif %}
```

ใน condition ของ if เราสามารถใส่ and, or หรือ not ได้ เช่น

```
{% if athlete_list and coach_list %}
    Both athletes and coaches are available.
{% endif %}

{% if not athlete_list %}
    There are no athletes.
{% endif %}

{% if athlete_list or coach_list %}
    There are some athletes or some coaches.
{% endif %}

{% if not athlete_list or coach_list %}
    There are no athletes or there are some coaches.
{% endif %}
```

และสามารถใช้ logic operations ต่างๆได้ ได้แก่

- `==`, `!=`

```
{% if somevar != "x" %}
  This appears if variable somevar does not equal the st
ring "x",
  or if somevar is not found in the context
{% endif %}
```

- `>`, `>=`, `<`, `<=`

```
{% if somevar <= 100 %}
  This appears if variable somevar is less than 100 or e
```

```
qual to 100.
{% endif %}
```

- `in`, `not in`

```
{% if user in users %}
  If users is a QuerySet, this will appear if user is an
  instance that belongs to the QuerySet.
{% endif %}
```

- `is`, `is not`

```
{% if somevar is True %}
  This appears if somevar is not True, or if somevar is
not found in the
  context.
{% endif %}

{% if somevar is not None %}
  This appears if and only if somevar is not None.
{% endif %}
```

- **now**

```
It is {% now "jS F Y H:i" %}
```

- **url**

```
{% url 'some-url-name' v1 v2 %}
```

ตัวอย่างเช่น

```
path("client/<int:id>/", app_views.client, name="app-vie
ws-client")
```

ใน template จะใช้ tag **url** ดังนี้

```
{% url 'app-views-client' client.id %}
```

Built-in filter reference

Ref

- **add**

```
{{ value|add:"2" }}
```

- **addslashes**

```
{{ value|addslashes }}
```

- **capfirst**
- **cut**

```
{{ value|cut:" " }}
```

ถ้าค่าของ `value` คือ "String with spaces" จะได้ output เป็น "Stringwithspaces"

date - ใช้ในการ format ข้อมูล datetime Ref

```
{{ value|date:"D d M Y" }} {{ value|time:"H:i" }}
```

- **default**

```
{{ value|default:"nothing" }}
```

ถ้าค่าของ `value` แปลงเป็น boolean ได้เป็น False จะแสดงค่า "nothing"

- **default_if_none**

```
{{ value|default_if_none:"nothing" }}
```

ถ้าค่าของ `value` เป็น None จะแสดงค่า "nothing"

- **dictsort**

```
{{ value|dictsort:"name" }}

<!-- ถ้าค่าของ value เป็น -->
[
    {"name": "zed", "age": 19},
    {"name": "amy", "age": 22},
    {"name": "joe", "age": 31},
]

<!-- จะได้ output -->
[
    {"name": "amy", "age": 22},
    {"name": "joe", "age": 31},
    {"name": "zed", "age": 19},
]
```

- **divisibleby**

```
{{ value|divisibleby:"3" }}
```

- **join**

```
{{ value|join:" // " }}
```

- **length**

```
{{ value|length }}
```

- **time**

```
{{ value|time:"TIME_FORMAT" }}
```

- **timesince**

```
{{ blog_date|timesince:comment_date }}
```

แสดงเวลาจากค่าตัวแปร (`blog_date`) จนถึงเวลา `now` (เช่น “4 days, 6 hours”)

รับ optional argument เป็นเวลาที่ต้องการเปรียบเทียบแทนค่า `now`

- **timeuntil**

```
{{ conference_date|timeuntil:from_date }}
```

คล้ายกับ "timesince" แต่จะวัดเวลาจาก `now` ไปจนถึงค่าตัวแปรซึ่งเป็นวันในอนาคต (`conference_date`)

รับ optional argument เป็นเวลาที่ต้องการเปรียบเทียบแทนค่า `now`

- **truncatechars**

```
{{ value|truncatechars:7 }}
```

ถ้ายาวเกินค่าที่กำหนดจะถูกแทนด้วย "..."

- **urlencode**

```
{{ value|urlencode }}
```

If value is "https://www.example.org/foo?a=b&c=d", the output will be "https%3A//www.example.org/foo%3Fa%3Db%26c%3Dd".

▼ Tutorial

WEEK 8: Django Templates

สำหรับการใช้งาน static files (images, css ,js เราจะต้องทำการตั้งค่ากันก่อน)

Configuring static files

1. ดูว่าใน "INSTALLED_APPS" นั้นมี "django.contrib.staticfiles" อยู่

2. เพิ่มการกำหนดว่า static files ของโปรเจคจะอยู่ที่ folder ไหนใน `settings.py`

```
STATIC_URL = "static/"

STATICFILES_DIRS = [
    BASE_DIR / "static",
]
```

1. สร้าง folder `static` ในระดับเดียวกับ folder `polls` และใส่ไฟล์ statics ที่ต้องใช้ใน folder `static`
2. ในไฟล์ template ของคุณ ให้ load template tag "static" และเรียกใช้ดังนี้

```
{% load static %}
<img src="{% static 'my_app/example.jpg' %}" alt="My ima
ge">
```

1. เก็บ static files ใน folder "static"

Let's start the tutorial

เราจะมาปรับแก้ไข app "polls" ที่เราได้ทำกันไปใน WEEK 7 กันนะครับ

index.html

ขั้นตอนแรกเราจะแก้ไขในไฟล์ `polls/views.py` เพื่อแสดงคำถามทั้งหมด

```
...
class IndexView(View):
    def get(self, request):
        question_list = Question.objects.order_by("-pub_
date")
        context = {"question_list": question_list}
        return render(request, "index.html", context)
...
```

จากนั้นเราจะมาปรับไฟล์ `/polls/templates/index.html`

```
<html>
    <head>
    </head>
    <body>
        <h1>Lastest questions</h1>
        {% if question_list %}
            <ul>
            {% for question in question_list %}
                <li><a href="/polls/{{ question.id }}/">
{{ question.question_text }}</a></li>
            {% endfor %}
            </ul>
        {% else %}
            <p>No polls are available.</p>
        {% endif %}
    </body>
</html>
```

โดยเพิ่มให้แสดงคำถาม `question_text` ไม่เกิน 40 ตัวอักษร truncatechars และ แสดง pub_date "Fri 31, Jan 24" date

```
...
<li><a href="/polls/{{ question.id }}/">{{ question.ques
tion_text|truncatechars:40 }}</a> - Published date: {{qu
estion.pub_date|date:"D d, M y"}}</li>
...
```

ผมคิดว่าเราน่าจะทำให้เวบไซต์ของเราสวยขึ้นสักหน่อยโดยใช้ css framework "Bulma"

ก่อนอื่นไป download ไฟล์ bulma.css มาจาก website "https://bulma.io/"

จากนั้นวางไฟล์ bulma.css ไว้ใน folder `static/css/bulma.css`

และเพิ่ม `{% load static %}` และ import css ไฟล์ใน <head></head>

```
{% load static %}
<!DOCTYPE html>
```

```
<html>
    <head>
        <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, i
nitial-scale=1">
    <title>Index</title>
    <link rel="stylesheet" href="{% static 'css/bulma.cs
s' %}">
    </head>
    <body>
        <section class="section">
            <h1 class="title">All questions</h1>
            <div class="content">
            {% if question_list %}
                <ul>
                {% for question in question_list %}
                <li><a href="{% url 'detail' question.id
%}">{{ question.question_text|truncatechars:40 }}</a>
                    - Published date: {{question.pub_dat
e|date:"D d, M y"}}</li>
                {% endfor %}
                </ul>
            {% else %}
                <p>No polls are available.</p>
            {% endif %}
            </div>
        </section>
        <section class="section">
            <p class="subtitle">
                Total: {{question_list|length}} question
s <!-- ใช้ filter length เพื่อแสดงขนาดของ list question_list
-->
            </p>
        </section>
```

```
    </body>
</html>
```

detail.html

จากนั้นเราไปแก้ไขหน้า `detail.html` ให้สวยงามกันบ้าง

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Detail</title>
    <link rel="stylesheet" href="{% static 'css/bulma.css' %}">
    </head>
    <body>
        <section class="section">
            <h1 class="title">Question: {{ question.question_text }}</h1>
            <p class="subtitle">Published date: {{ question.pub_date|date:"D d, M Y" }} ({{ question.pub_date|timesince }})</p>
            <div class="content">
            <ul>
                {% for choice in choices %}
                    <li>Choice {{forloop.counter}} => {{ choice.choice_text }} (Votes: {{choice.votes}})</li>
                {% endfor %}
            </ul>
            <a href="{% url 'vote' question.id %}" class="button ml-3">Let's Vote</a>
            <p class="subtitle is-6">Current time: {% now "D d, M Y" %}</p>
```

```
            </div>
        </section>
    </body>
</html>
```

จะเห็นได้ว่าเรามีการใช้งาน filter `timesince` และ ใช้งาน `forloop.counter` เพื่อแสดงหมายเลขรอบของ for loop

และมีการใช้งาน template tag `now` เพื่อแสดง วัน-เวลา ปัจจุบัน

vote.html

สุดท้ายเราไปแก้ไขหน้า `vote.html` ให้สวยงามกัน

```
{% load static %}
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, i
nitial-scale=1">
    <title>Detail</title>
    <link rel="stylesheet" href="{% static 'css/bulma.cs
s' %}">
    </head>
    <body>
        <section class="section">
            <h1 class="title">Question: {{ question.ques
tion_text }}</h1>
            <p class="subtitle">Published date: {{ quest
ion.pub_date|date:"D d, M Y" }} ({{ question.pub_date|ti
mesince }})</p>
        </section>
        <div class="container pl-6 pt-1">
            <form action="{% url 'vote' question.id %}"
method="POST">
            {% csrf_token %}
```

```
            <div class="field">
                {% for choice in choices %}
                <div class="control">
                    <label class="radio">
                        <input type="radio" id="choice{{
forloop.counter }}" name="choice" value="{{ choice.id
}}">
                        {{ choice.choice_text }}
                    </label>
                </div>
                {% endfor %}
            </div>

            <div class="field">
                <div class="control">
                  <input type="submit" class="button is-
link" value="Submit">
                </div>
            </div>
            </form>
        </div>
    </body>
</html>
```

จะเห็นได้ว่ามีการใช้งาน template tag `csrf_token` และ `url`

What is CSRF Token in Django?

Source

> Django provides a feature to prevent such types of malicious attacks. When a user is authenticated and surfing on the website, Django generates a unique CSRF token for each session. This token is included in forms or requests sent by the user and is checked by the server to

> verify that the request is coming from the authenticated user and not from a malicious source.

จะแนบไฟล์

- Jupyter notebook week 4
- Jupyter notebook week 5
- Project week 7, 8