day03-虛擬環境、Poetry與第一個Django專案

在我們建構第一個Django專案之前，必須要先聲明虛擬環境的重要性

以前還沒什麼觀念的時候，看到一些影片的安裝教學都直接安裝軟體，安裝的方式可能也有百百種

等到之後想移除時，可能在多個路徑下都有不同版本的相同軟體

在使用IDE引用來源時可能也是胡亂使用，這時候就很想要重頭開始一個乾淨的環境

也體現出使用虛擬環境的重要性

python有以下幾種套件可以幫忙建立虛擬環境

1. **venv**
    - Python 3.x內建的模組，無需額外安裝。使用`python -m venv <環境名稱>`命令可以輕鬆建立虛擬環境
    - 只能使用創建時的python版本，且無法指定系統中不存在的python版本
    - 不能查看虛擬環境的環境列表
2. **virtualenv**
    - 可以使用`pip install virtualenv`安裝，然後使用`virtualenv <環境名稱>`來創建虛擬環境。它支持Python 2和Python 3，並且能夠創建不同Python版本的虛擬環境
    - 跟venv功能雷同，區別在存放虛擬環境的位置
3. **pipenv**
    - 結合了`pip`和`virtualenv`的功能，提供了更高層次的包管理和虛擬環境管理。使用`pipenv install`可以創建虛擬環境並安裝依賴。適合需要管理多個依賴的項目
    - 專案的前景較為堪憂
4. **Poetry**

- 一個較新的包管理和虛擬環境管理工具，專注於簡化Python專案的依賴管理。使用`poetry new <專案名稱>`創建新專案時會自動建立虛擬環境，並且提供了清晰的配置文件管理
- 學習曲線較陡峭

這些工具各有特點，看各位自己的習慣進行使用，在這個系列文章中，我會使用poetry來建構我的虛擬環境。如果想更了解一些poetry的指令，我之前有寫過一篇文章就不詳細展開了

https://ithelp.ithome.com.tw/articles/10340858

建構環境

```
# 初始化虛擬環境
poetry new django_project

# 安裝Django 這邊使用4.2LTS版本，也可以直接選擇最新的安裝
poetry add Django==4.2
```

確認是否成功，除了可以看`poetry.lock`檔案之外，輸入指令`poetry show --tree`也可以看安裝套件的相關依賴

```
django 4.2 A high-level Python web framework that encourages rap
├── asgiref >=3.6.0,<4
├── sqlparse >=0.3.1
└── tzdata *
```

第一個Django專案

那我們現在就來進到我們第一個專案～

```
# 先刪除不必要的包
rm -rf django_project

# 啟動虛擬環境
poetry shell

# 建立專案
django-admin startproject django_project

# 查看專案結構
tree
```

```
.
├── django_project
│   ├── __init__.py
│   ├── asgi.py # 如果專案需要用到非同步功能，使用asgi啟動django專案
│   ├── settings.py # 重要的設定都包含在裡面
│   ├── urls.py # 路由，說明哪些url指到哪些view中
│   └── wsgi.py # 如果不使用非同步，會使用wsgi啟動Django專案
└── manage.py # 命令工具，可以執行多種功能
```

我們可以透過`python3 manage.py`指令來看有哪些命令可以執行

```
# 這邊只列出幾個我比較常使用的
[auth]
    createsuperuser # 用來建立超級用戶 管理後台與相關權限

[django]
        dumpdata # 匯出資料庫的資料
    inspectdb # 匯出目前DB中所有table的配置
    loaddata # 將外部資料載入資料庫
    makemigrations # 整理出model有變動的部分
    migrate # 將makemigrations紀錄的資料實施到DB中
```

```
    shell # 進入shell模式，可以做一些基本的CRUD操作
    startapp # 建立新的app
    test # 對每個app下的test*.py檔中的內容進行測試

[staticfiles]
    collectstatic # 將每個app下以及安裝套件下的所有靜態資源根據參數匯入到
    runserver # 啟動本機專案
```

現在對這些命令如果還沒有什麼概念的話也沒有關係，之後操作到的時候就會有想法了

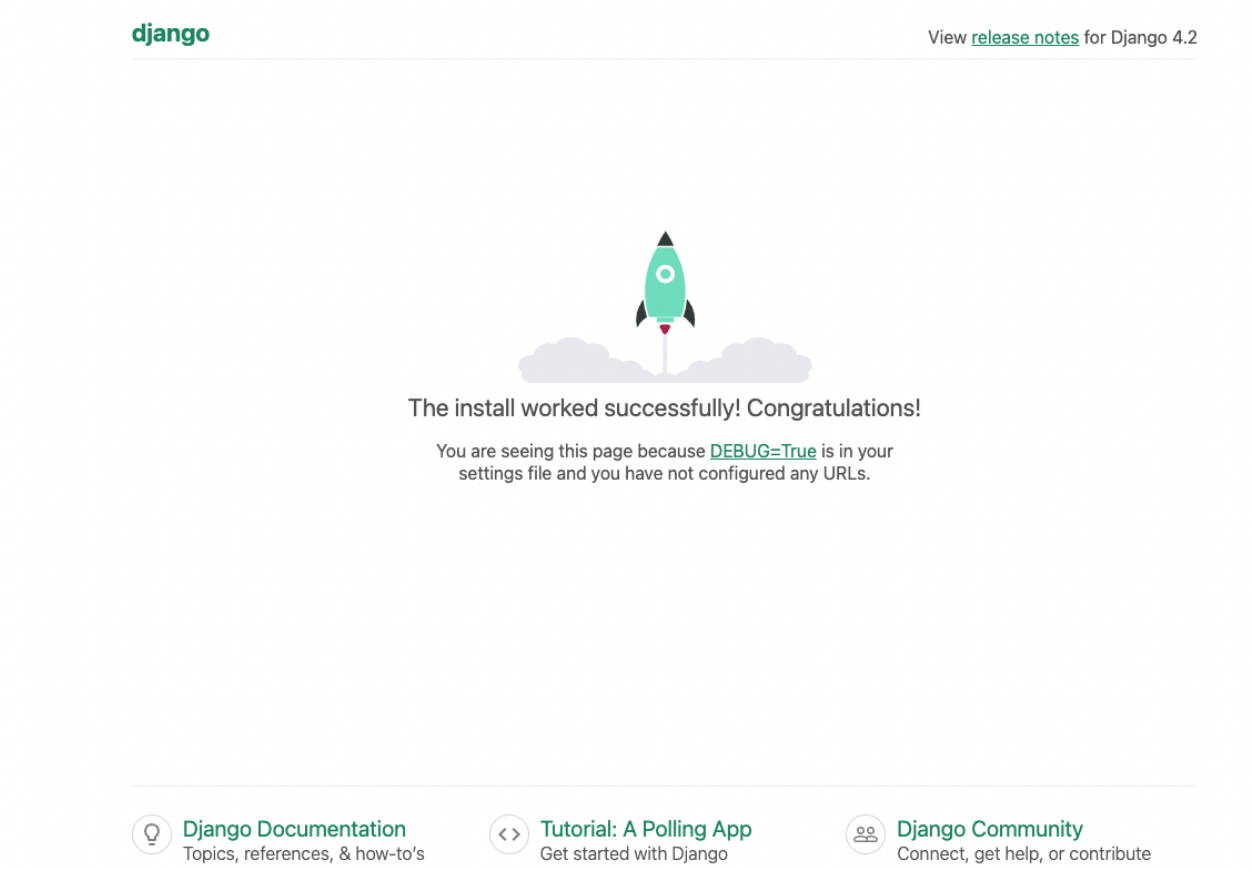
並且除了Django預設的命令，也可以建立`management/commands`來客製化自己的指令

那就來啟動我們的專案吧

```
python3 manage.py runserver
# 可以在最後面加上port ex: python3 manage.py runserver 8888
# 默認啟動在8000 port上
```

接著在瀏覽器輸入`http://127.0.0.1:8000/`

能看到以下畫面就代表我們成功啟動Django專案了！



基礎設置

我們先在manage.py同目錄建立其他幾個資料夾static, staticfiles與templates以及建立.env檔

專案目錄如下：

```
.
├── django_project
│   ├── __init__.py
│   ├── __pycache__
│   │   ├── __init__.cpython-312.pyc
│   │   └── settings.cpython-312.pyc
│   ├── asgi.py
```

```
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── manage.py # .env跟他同級
├── static # 放置靜態資源
│   ├── css
│   ├── images
│   └── js
├── staticfiles # 之後方便整合nginx等找到所有的靜態資源 之後會再展開說明
└── templates # 放網頁模板
```

.env中除了配置資料庫的設定值之外，還可以把許多敏感的資訊放進來

```
# .env
DB_HOST=127.0.0.1
DB_USER=your_db_user
DB_PWD=your_db_password
DB_NAME=your_db_name

SECRET_KEY=xxxxx

DEBUG=false
```

還需要安裝相關套件，才能順利導入環境變數

```
poetry add load-dotenv
```

接著到settings.py中導入相關的環境變數，並且簡單介紹基本的配置

```
# settings.py

# 環境變數檔案的路徑
```

```
from dotenv import load_dotenv
BASE_DIR = Path(__file__).resolve().parent.parent
ENV_FILE_PATH = BASE_DIR.parent / '.env'

# 載入.env檔案
load_dotenv(ENV_FILE_PATH)

# 配置變數
# postgreSQL 相關設定
DB_HOST = os.getenv("DB_HOST")
DB_USER = os.getenv("DB_USER")
DB_PWD = os.getenv("DB_PWD")
DB_NAME = os.getenv("DB_NAME")

# SECURITY WARNING: keep the secret key used in production secre
SECRET_KEY = os.getenv("SECRET_KEY")

# SECURITY WARNING: don't run with debug turned on in productior
DEBUG = os.getenv("DEBUG")

# 註冊專案下的所有APP
INSTALLED_APPS = [
    'django.contrib.admin',
        ....
    'django.contrib.staticfiles',
]

# 配置中間件
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
        ....
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

# 配置template的位置
TEMPLATES = [
```

```
    {
        'BACKEND': 'django.template.backends.django.DjangoTempla
        'DIRS': [os.path.join(BASE_DIR, 'templates/')], # 根據你t
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.mess
            ],
        },
    },
]

# 連接DB 可以配置多個DB，但是一定要有一個default
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': DB_NAME,
        'USER': DB_USER,
        'PASSWORD': DB_PWD,
        'HOST': DB_HOST,
        'PORT': '5432',
    },
}

# 配置靜態資源
STATIC_URL = 'static/' # 靜態資源的url
STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')] # 在dev環境
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles') # 用在produc

# 配置時區
TIME_ZONE = 'Asia/Taipei'

# 配置上傳檔案的路徑
```

```
MEDIA_URL = '/media/' # 上傳檔案的url配置
MEDIA_ROOT = os.path.join(BASE_DIR, 'uploads') # 讓django去這裡找
```

這邊不詳細介紹static跟staticfile應用上的區別，之後如果需要用到的時候再來說明

此外因為在後面的專案會需要跟據schema來區分用戶的資料，所以DB推薦使用postgreSQL來進行配置，我是直接使用Docker啟動，相關配置如下

```
image: postgres:13-alpine
volumes:
    - ../postgres_data:/var/lib/postgresql/data
ports:
    - "5432:5432"
environment:
    - POSTGRES_DB=${DB_NAME}
    - POSTGRES_USER=${DB_USER}
    - POSTGRES_PASSWORD=${DB_PWD}
```

初入Django MTV

那我們現在再來複習一下Django的基本架構

使用者根據特定`url`發請請求→`url`找到相對應的`View`→`View`依據特定邏輯處理`Model`中的資料→`View`拿到最終數據後渲染到`Templates`上

作圖

接著我們就來透過實作來完成這樣的流程吧～

首先建立article app

```
python3 manage.py startapp article
```

然後在settings.py中註冊app

```
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    # my apps
    "article",
]
```

進入app下的models.py

在models.py中，我們配置需要在資料庫中建立的表格，一個類別相當於一個表格

每一個類別都需要繼承`models.Model`，而類別的屬性則為表格中的欄位

我們需要新增的表格如下：

```
# article.models.py
from django.db import models

class Article(models.Model):
    article_id = models.AutoField(primary_key=True)
    title = models.CharField(max_length=120)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey("Category", on_delete=models.CA
    author = models.ForeignKey("Author", on_delete=models.CASCAD
    tags = models.ManyToManyField("Tag")


class Category(models.Model):
    category_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=120)
```

```
class Author(models.Model):
    author_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=120)
    age = models.IntegerField()


class Tag(models.Model):
    tag_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=120)
```

我們可以看到屬性也是使用`models`模塊下不同的類別，設定欄位的類型之外，也進行其他的配置

例如：

- AutoField(primary_key=True)：表示唯一標識符。AutoField 會自動增加，`primary_key=True` 表示這是主鍵，通常會被映射為資料庫中的 INTEGER 類型，
- CharField(max_length=120)：CharField 用於存儲較短的字符串，`max_length=120` 限制最大長度為120個字符，被映射為 VARCHAR類型
- TextField()：TextField 用於存儲長文本，沒有長度限制，被映射為TEXT 類型
- DateTimeField()：創建時間，`auto_now_add=True`使其在創建時自動設置為當前時間，而`auto_now=True` 使其在每次保存時自動更新為當前時間。被映射為 DATETIME 類型
- ForeignKey("Category", on_delete=models.CASCADE)：與 Category 模型的外鍵關係，CASCADE 表示當關聯的 Category 被刪除時，文章也會被刪除
- ManyToManyField("Tag")：與 Tag 模型的多對多關係，會建立一個額外的關聯表，包含兩個外鍵欄位，分別指向相關的兩個表

其他更詳細的配置可以看官方文檔：

https://docs.djangoproject.com/en/4.2/ref/models/fields/#field-types

接下來就要將這個設定遷移到資料庫中

設置好我們的model後，要將其遷移到我們的資料庫中

參考：

https://vocus.cc/article/64fec089fd897800018b21c1

https://www.cnblogs.com/doublexi/p/15783355.html

https://www.djangoproject.com/download/

https://www.geeksforgeeks.org/custom-django-management-commands/