Day07: Django in 2024: 強大的Django Admin

今天終於進到Django Admin了，也是Django提供的管理介面俗稱後台。主要是根據定義好的Model做出以模型為中心的介面，讓使用者能夠快速修改資訊，除了預設的介面之外，Django還是保留相當多的彈性在上面，設計好模板路徑便能改寫對應的頁面，並且也能客製化權限，搭配昨天介紹的表單，快速搭建相關邏輯的同時還是有辦法處理複雜的邏輯

今日的重點如下：

- Django admin初探
- 註冊模型，迅速執行CRUD
- 設置名稱，後台使用更便利

程式碼：https://github.com/class83108/django_project/tree/admin

Django admin初探

這些Django用來建立後台的配置，是一開始startproject時就會設置好不需要調整

```
# settings.py

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

```
    "django.contrib.messages",
]

MIDDLEWARE = [
    ...
    "django.contrib.sessions.middleware.SessionMiddleware",
        ...
    "django.contrib.auth.middleware.AuthenticationMiddleware",
    "django.contrib.messages.middleware.MessageMiddleware",
        ...
]

TEMPLATES = [
    {
                ...
        "OPTIONS": {
            "context_processors": [
                                ...
                "django.template.context_processors.request",
                "django.contrib.auth.context_processors.auth",
                "django.contrib.messages.context_processors.mess
            ],
        },
    },
]
```

進入後台需要先建立用戶，透過指令來建立超級用戶

```
python3 manage.py createsuperuser
```

依照提示來完成

```
Username (leave blank to use 'xxx'): admin
Email address:
Password:
Password (again):
Error: Your passwords didn't match. # 如果密碼打錯不用擔心會驗證
Password:
Password (again):
Superuser created successfully.
```

這個用戶不用做任何設定就有對於所有model的權限，並且也有staff的權限

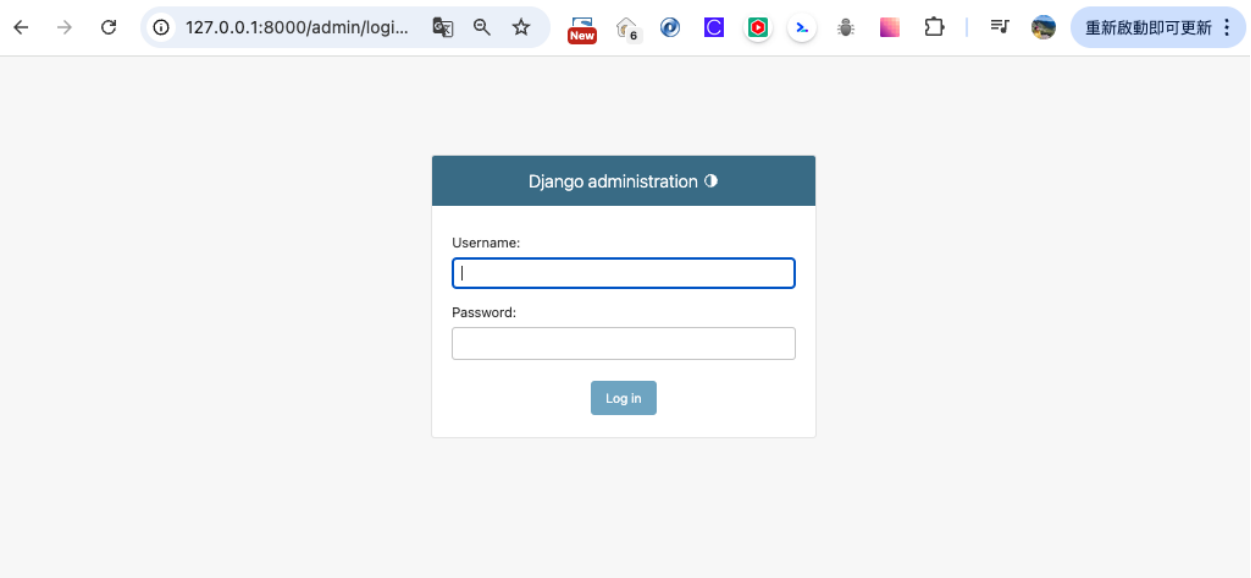
有staff權限的人才能進入admin後臺

接著我們去確認一下路由

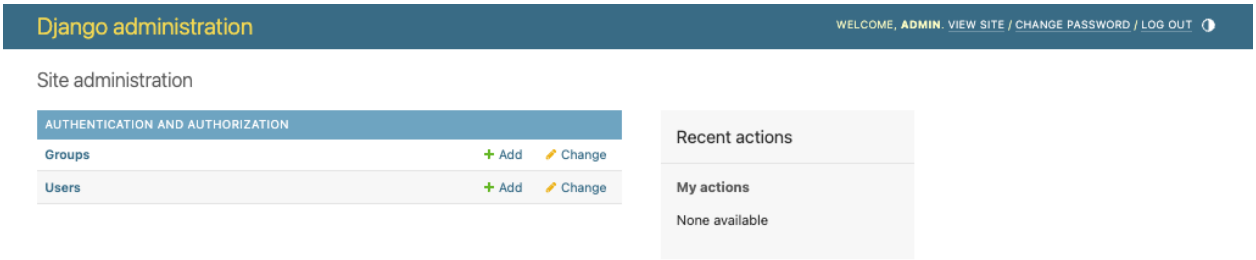
```
# 根目錄下的urls.py

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

可以看到Django已經有預設好的配置，這也代表我們如果之後想要調整路由也是能夠自己定義

輸入網址可以看到登入畫面

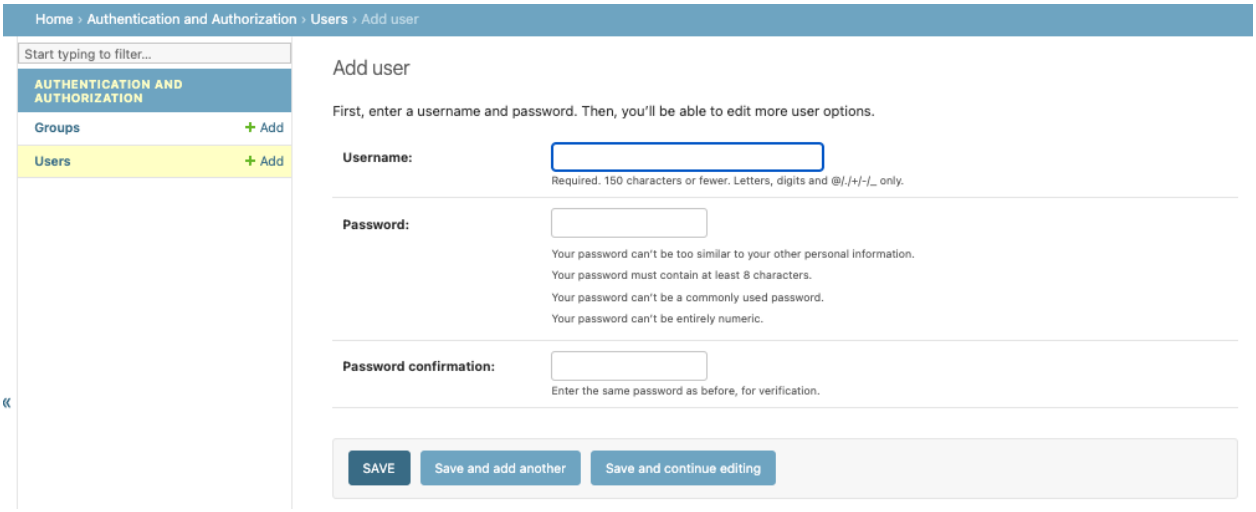


輸入剛剛設置的帳密後進行登入

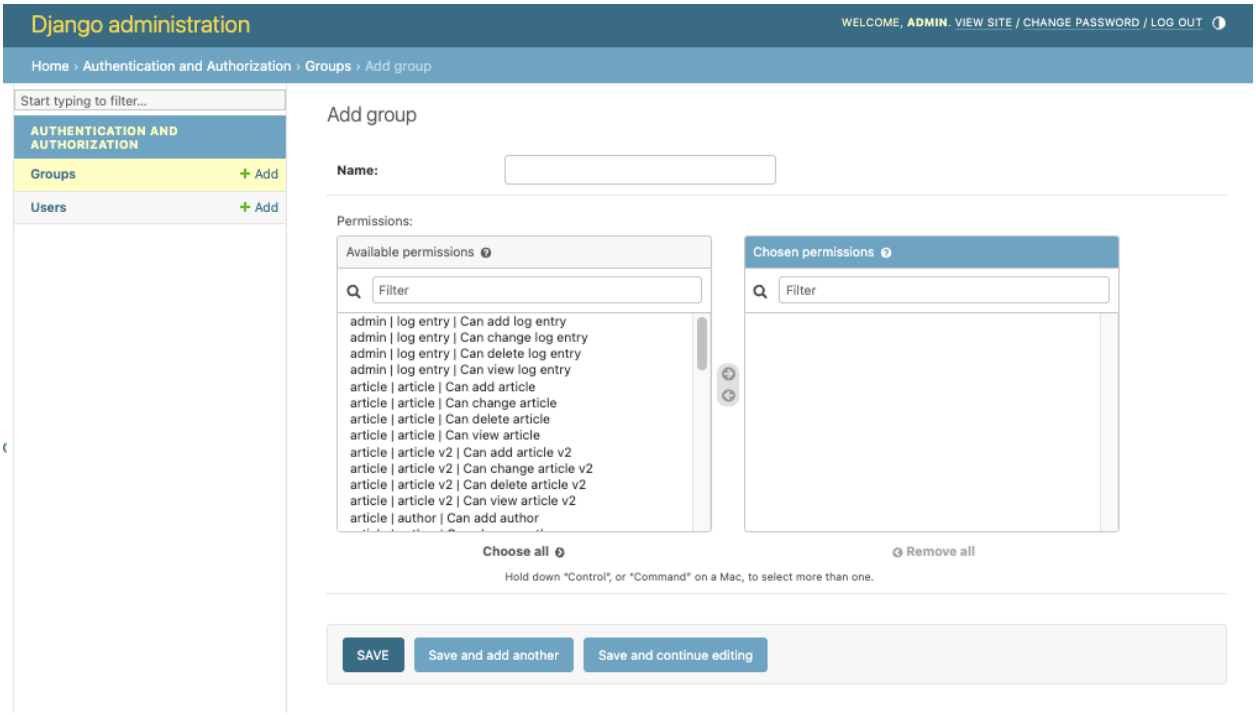


在還沒做任何設置的情況下，可以看到header跟content分別有不同的資訊

- header中除了顯示當前用戶外，其中也有設置好修改密碼跟登出的指示
- Users也就是管理後台的User模型，可以透過介面直接添加或是修改



- Group則是方便賦予不同的user設置好的權限，不需要一個一個user慢慢設置

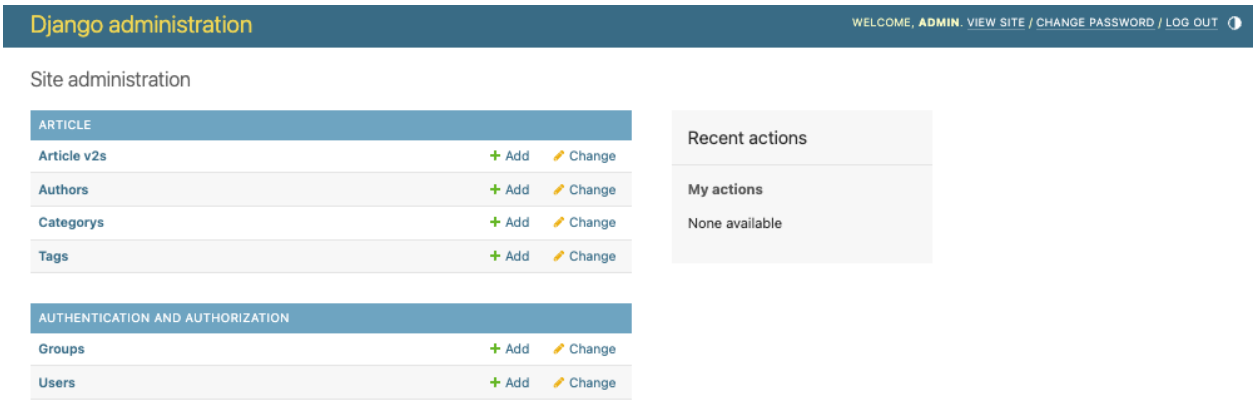


註冊模型，迅速執行CRUD

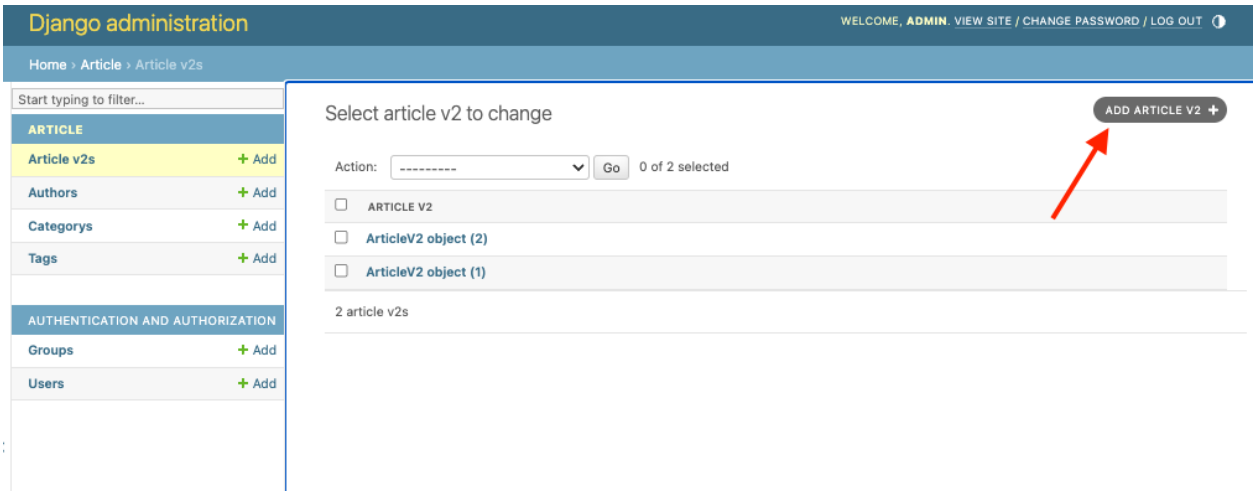
從首頁可以看到現在還沒有任何一個我們之前的model，因此我們需要進行註冊

```
# app下的admin.py
from django.contrib import admin

from .models import ArticleV2, Tag, Category, Author

admin.site.register(ArticleV2)
admin.site.register(Tag)
admin.site.register(Category)
admin.site.register(Author)
```

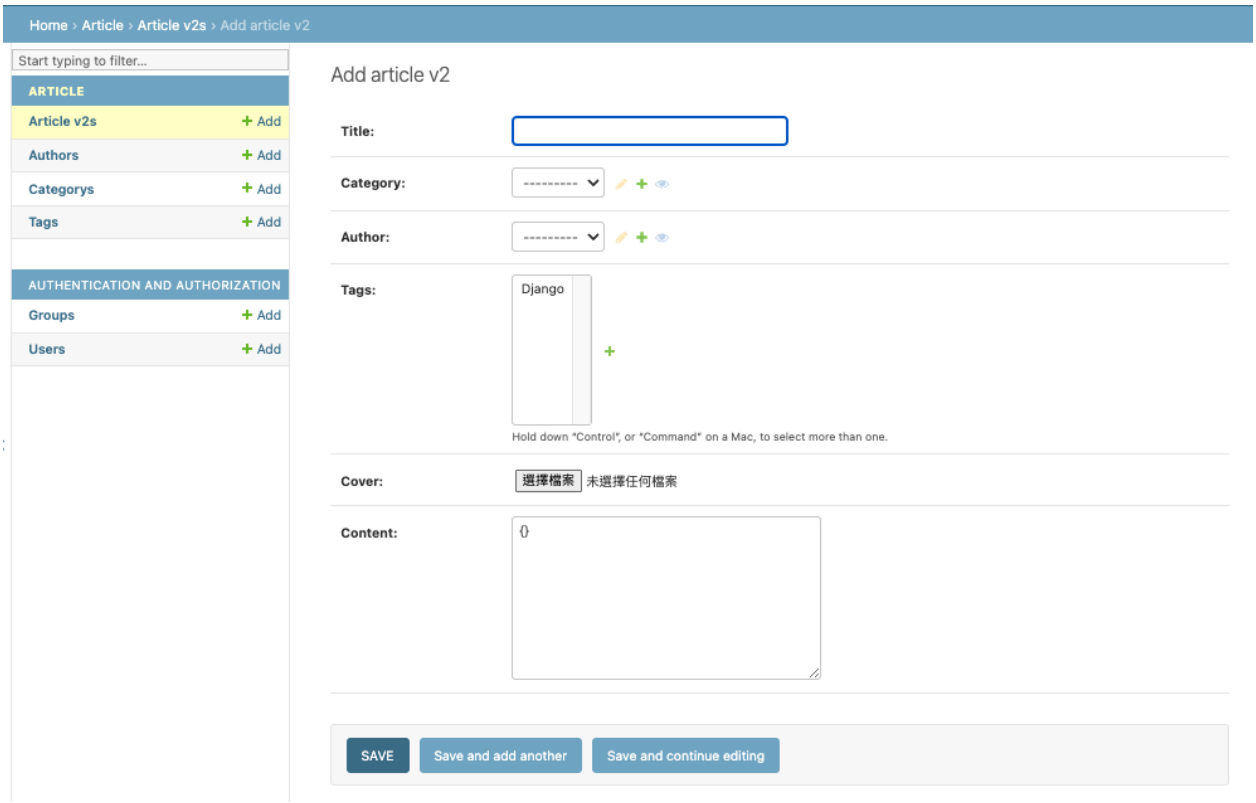
重新整理我們的畫面後，就可以看到



我們先進入ArticleV2的列表頁，同時也可以看到右上角的添加按鈕


Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Article › Article v2s
Start typing to filter...
ARTICLE
Article v2s + Add
Authors + Add
Categorys + Add
Tags + Add
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
Select article v2 to change
ADD ARTICLE V2 +
Action: --------- Go 0 of 2 selected
ARTICLE V2
ArticleV2 object (2)
ArticleV2 object (1)
2 article v2s

點擊後可以看到

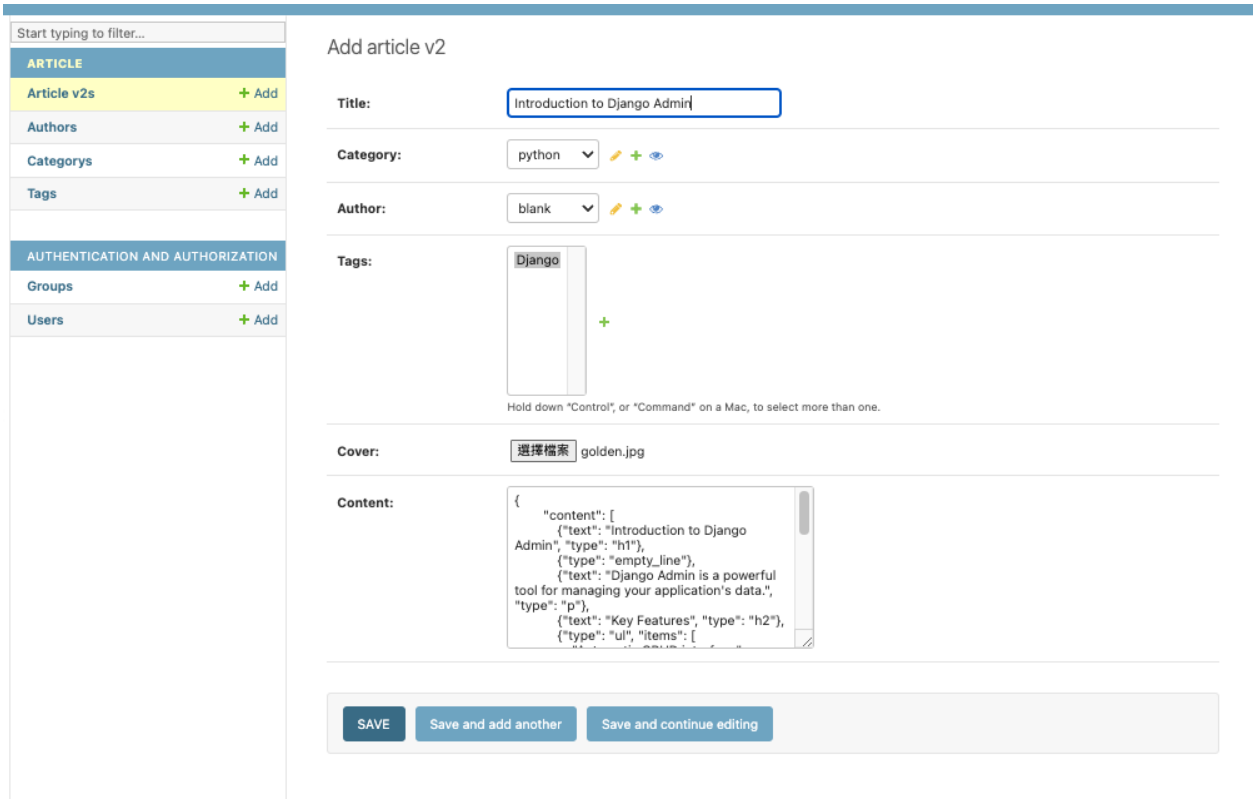

Home › Article › Article v2s › Add article v2
Start typing to filter...
ARTICLE
Article v2s + Add
Authors + Add
Categorys + Add
Tags + Add
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
Add article v2
Title:
Category: ---------
Author: ---------
Tags: Django
Hold down "Control", or "Command" on a Mac, to select more than one.
Cover: 選擇檔案 未選擇任何檔案
Content: {}
SAVE
Save and add another
Save and continue editing

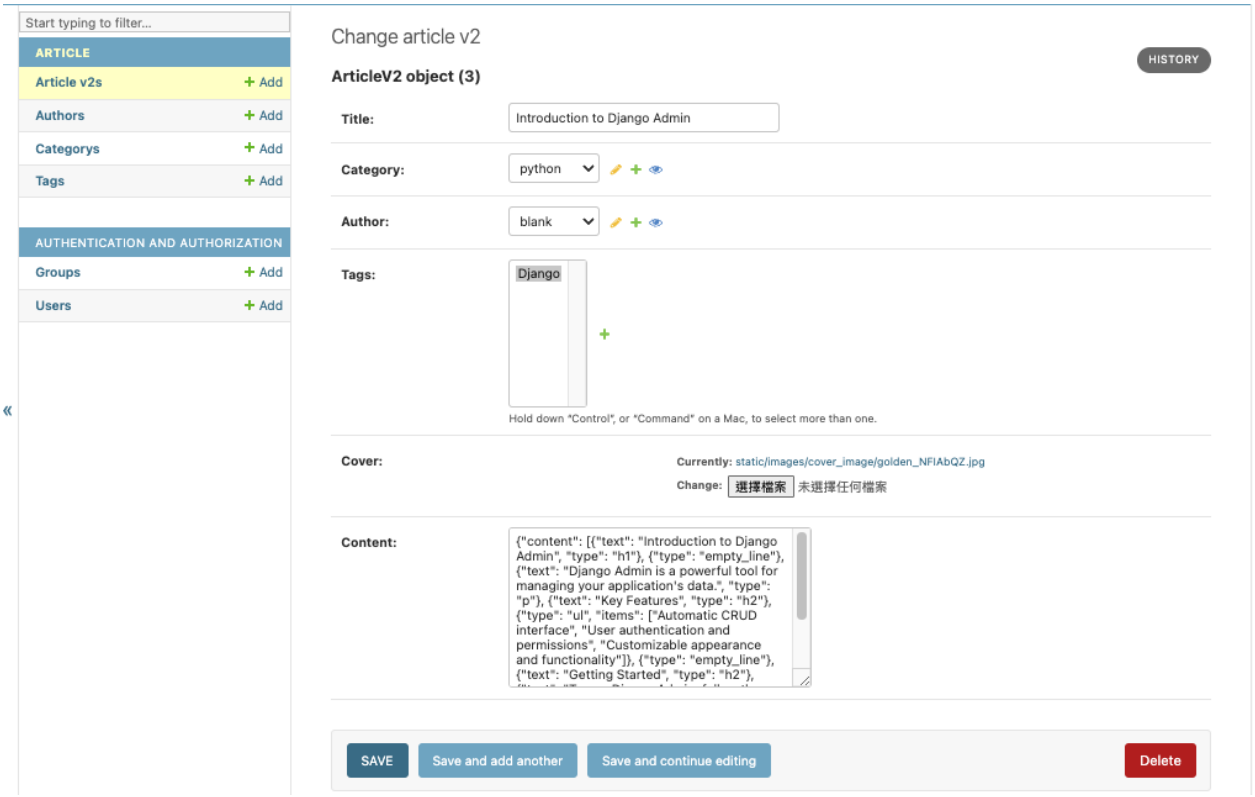
在輸入資料以前，因為我們的content是打算是用markdown語法，因此要自己突然生出資料可能比較麻煩，AI在做這件事情上就真的很方便，我是使用claude

```
# 我的input

class ArticleV2(models.Model):
    article_id = models.AutoField(primary_key=True)
    title = models.CharField(
        max_length=120, verbose_name="Title", unique=True, null=
    )
    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")

    # 新增欄位
    cover = models.ImageField(upload_to="static/images/cover_ima
    # 修改欄位
    content = JSONField(default=dict)

    class Meta:
        app_label = "article"
以上為我的model，幫我生出1筆資料有關django admin的資料
然後content是要類似這樣的JSON資料
"content": {            "content": [                {"text": "

# claude的output
from django.utils import timezone
from article.models import ArticleV2, Category, Author, Tag

# 假設這些相關模型已經存在，如果沒有，你需要先創建它們
category, _ = Category.objects.get_or_create(name="Django")
author, _ = Author.objects.get_or_create(name="Django Team")
tag, _ = Tag.objects.get_or_create(name="Admin")
```

```
article = ArticleV2.objects.create(
    title="Introduction to Django Admin",
    category=category,
    author=author,
    content={
        "content": [
            {"text": "Introduction to Django Admin", "type": "h1
            {"type": "empty_line"},
            {"text": "Django Admin is a powerful tool for manag
            {"text": "Key Features", "type": "h2"},
            {"type": "ul", "items": [
                "Automatic CRUD interface",
                "User authentication and permissions",
                "Customizable appearance and functionality"
            ]},
            {"type": "empty_line"},
            {"text": "Getting Started", "type": "h2"},
            {"text": "To use Django Admin, follow these steps:",
            {"type": "ol", "items": [
                "Create a superuser account",
                "Register your models in admin.py",
                "Customize the admin interface as needed"
            ]}
        ]
    }
)

# 添加標籤
article.tags.add(tag)

# 如果你有封面圖片，你可以這樣添加：
# from django.core.files import File
# article.cover.save('django_admin_cover.jpg', File(open('path/t

print(f"Created article: {article.title} (ID: {article.article_
```

那我們這邊就把他給的資料填入





可以看到儲存後我們也能馬上進行編輯或刪除

而後台預設還有一點十分方便的就是有關ManyToMany的資料能夠在當前頁面進行處理

點擊Tags旁邊的綠色加號，就會彈出視窗讓人新增，並且父視窗也會馬上出現該選項

如果想要做批量的操作，也能在列表頁進行操作

可以看到只要在admin.py中註冊模型，就能快速的建立起該Model的CRUD操作

是不是非常快速且方便呢？

設置名稱，後台使用更便利

從剛剛的範例中也可以觀察到，admin預設的許多名稱表示不夠直觀，我們可以透過不同的方式來進行調整

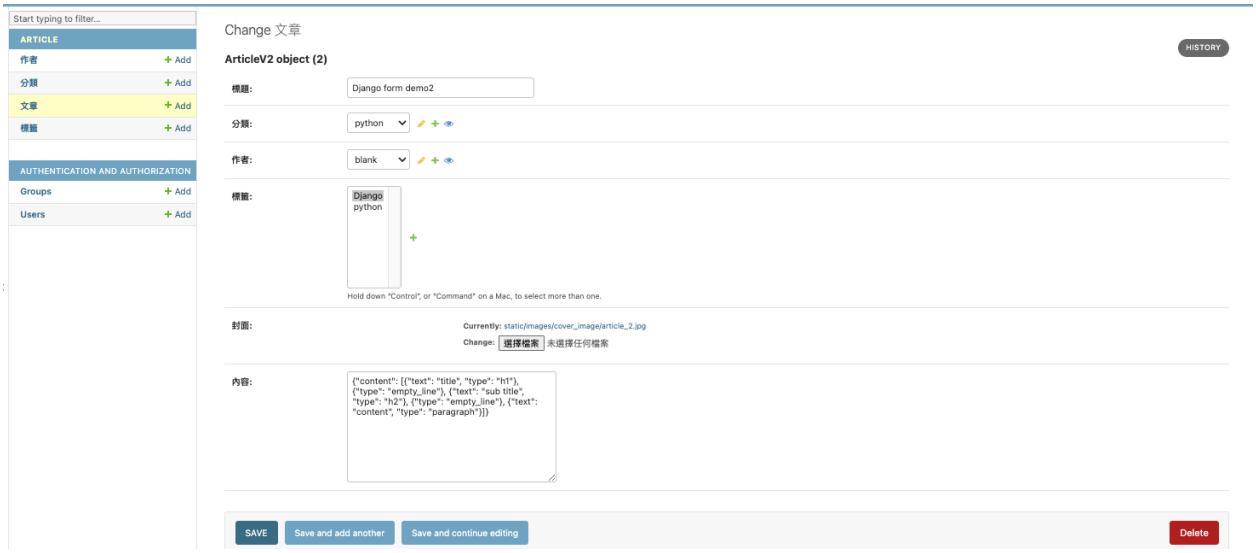
1. 可以直接修改models的配置

```
class ArticleV2(models.Model):
    article_id = models.AutoField(primary_key=True)
    title = models.CharField(
        max_length=120,
        unique=True,
        null=False,
        verbose_name="標題",
    )
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    category = models.ForeignKey(
        "Category", on_delete=models.CASCADE, verbose_name="分類
    )
    author = models.ForeignKey("Author", on_delete=models.CASCAD
    tags = models.ManyToManyField("Tag", verbose_name="標籤")

    # 新增欄位
    cover = models.ImageField(
        upload_to="static/images/cover_image", null=True, verbos
    )
    # 修改欄位
    content = JSONField(default=dict, verbose_name="內容")

    class Meta:
        app_label = "article"
```

```
        verbose_name = "文章"
        verbose_name_plural = "文章"


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

    def __str__(self) -> str:
        return self.name

    class Meta:
        app_label = "article"
        verbose_name = "分類"
        verbose_name_plural = "分類"


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

    def __str__(self) -> str:
        return self.name

    class Meta:
        app_label = "article"
        verbose_name = "作者"
        verbose_name_plural = "作者"


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

    def __str__(self) -> str:
```

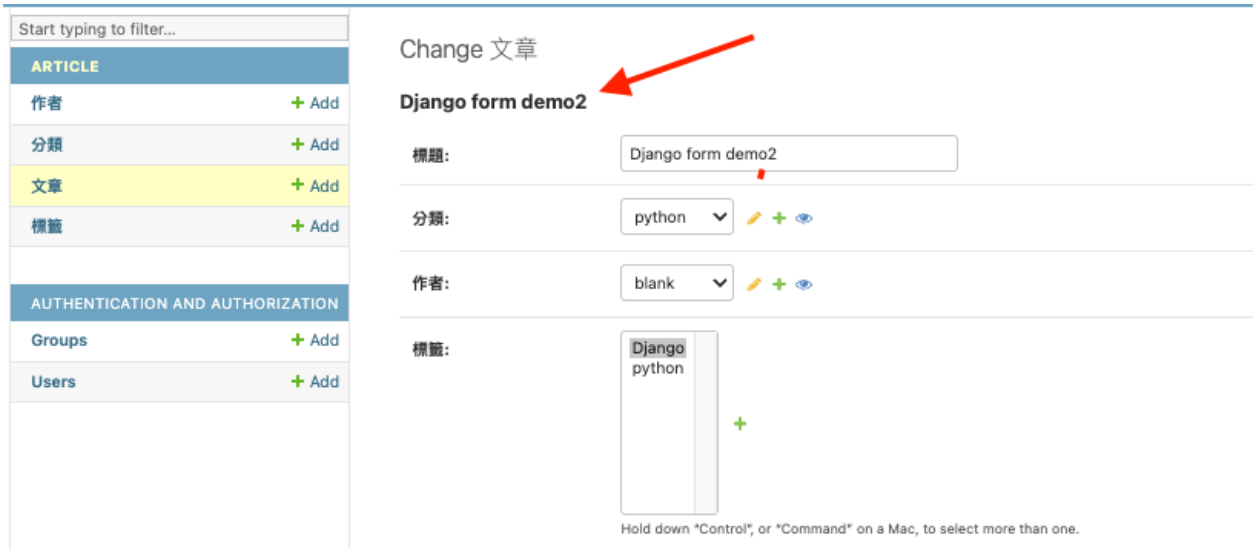
```
        return self.name

    class Meta:
        app_label = "article"
        verbose_name = "標籤"
        verbose_name_plural = "標籤"
```

因為有可能英文對於營運人員可能不容易理解，這邊示範時就直接改成中文，其中`verbose_name_plural`表示複數形式的名稱



可以看到model名稱與欄位名稱的確都進行了修改，但是還是有一些英文不是那麼直觀，可以繼續在model中進行修改

```
class ArticleV2(models.Model):
    ...

    def __str__(self) -> str:
        return self.title
```



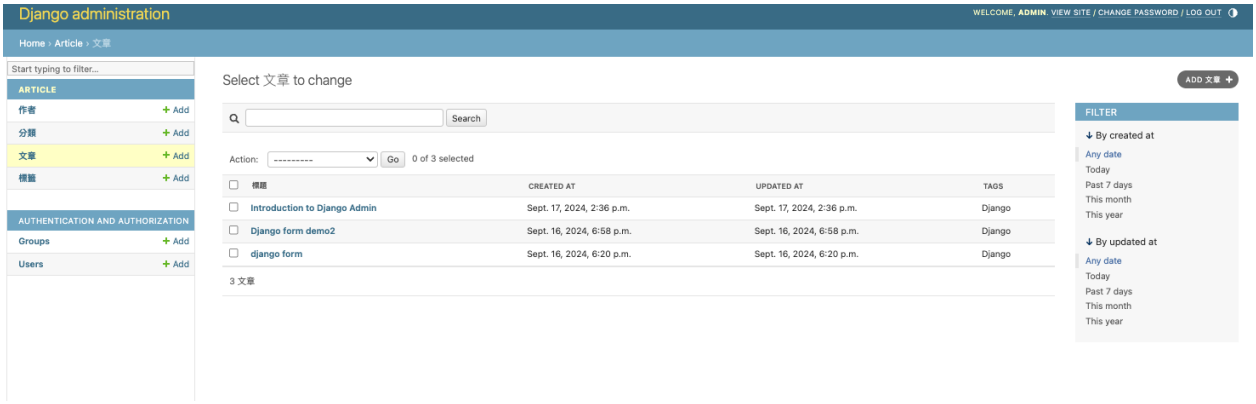
可以看到的確有進行修改了

那針對Admin的其他操作，我們還能做什麼修改呢？

1. 調整列表頁的相關欄位

```
# admin.py
class ArticleAdmin(admin.ModelAdmin):

    # 在列表頁顯示的欄位
    list_display = ["title", "created_at", "updated_at"]

    # 在列表頁可以搜尋的欄位
    search_fields = ["title"]

    # 在列表頁可以篩選的欄位
    list_filter = ["created_at", "updated_at"]

    def get_tags(self, obj):
        return ", ".join([tag.name for tag in obj.tags.all()])

    get_tags.short_description = "Tags"
    # 添加自定義的欄位
```

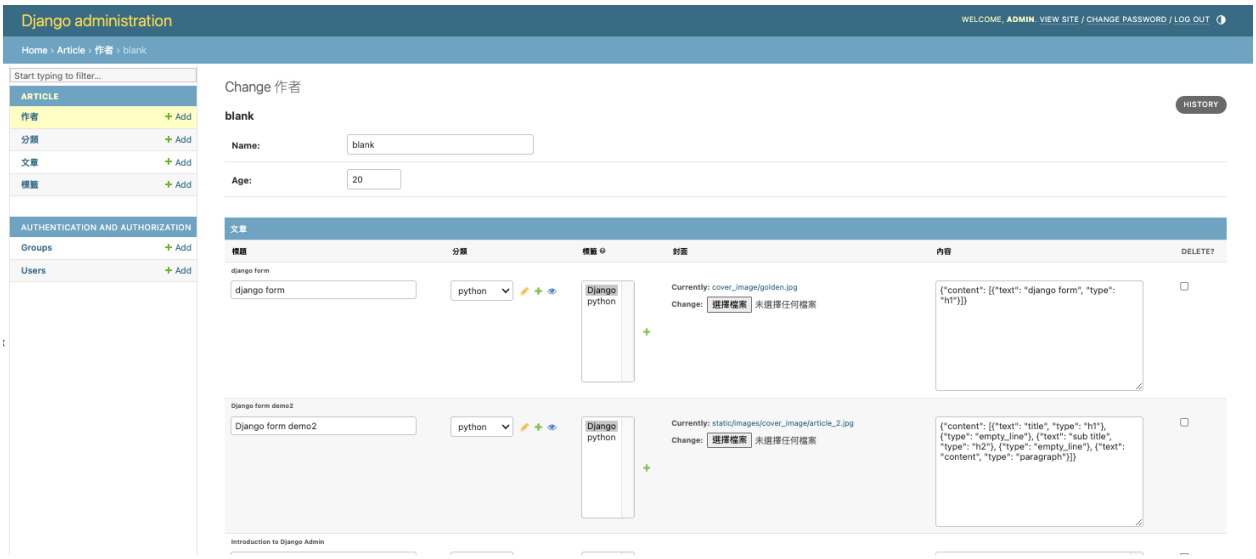
```
    list_display += ["get_tags"]

admin.site.register(ArticleV2, ArticleAdmin)
```



可以看到我們也能透過ORM語法來讓列表頁顯示其他關聯表格的相關資料

2. 在具有外鍵關聯的資料結構中，我們可以在修改頁面中直接編輯一對多關係中的"多"方模型

```
class ArticleInline(admin.TabularInline):
    model = ArticleV2
    extra = 1

class AuthorAdmin(admin.ModelAdmin):
    inlines = [ArticleInline]
    list_display = ["name", "age"]

admin.site.register(Author, AuthorAdmin)
```



今日總結

我們第一次使用Django Admin並且將我們之前建立的表格註冊進後台，並且透過

1. 調整Model本身的元數據
2. 修改admin.py中每個註冊model的方法或是屬性

來打造更加符合營運人員操作的後台，但是有關後台的操作遠不止如此，明天會更深入的介紹還有哪些是我們可以修改的設定，來打造屬於我們的後台系統