

# day22: Django REST framework: API最後一哩路-生成API文件

雖然我們的API都建立好了，但是即使是在路由都做好命名，且視圖中也都寫好註解，但是如果自己過了一段時間再回頭看，又或是今天多人開發下每個人的命名風格還是有些微的差異，需要能讀懂這些程式碼可能還是一項不小的工程。此外如果API有經過修改或是調整，同時也需要進行API文件的更新，此時也容易導致文件與API本身是不同步的情形，也因此透過自動化的方式建立起符合特定規範的API文件也是相當重要，且不可或缺的一步

今日重點：

- DRF內建支援生成OpenAPI格式文檔（已棄用）
- OpenAPI 3 vs Swagger/OpenAPI 2
- drf-spectacular
  - 基礎使用
  - 客製化API文件

## DRF內建支援生成OpenAPI格式文檔（已棄用）

Django REST framework（DRF）在先前的版本可以透過安裝 `coreapi` 套件來建立API文檔。但是根據官方文檔現在已經不在支援，官方更推薦使用其他第三方庫做使用，例如：[drf-spectacular](#)與[drf-yasg](#)

```
# 安裝套件
poetry add coreapi

# settings.py
REST_FRAMEWORK = {
    "DEFAULT_SCHEMA_CLASS": "rest_framework.schemas.coreapi.Auto

}

# urls.py
from rest_framework.documentation import include_docs_urls
urlpatterns = [
    path("docs/", include_docs_urls(title="DRF Demo API")),
]
```

## OpenAPI 3 vs Swagger/OpenAPI 2

我們上面有提到DRF

官方文檔推薦使用**drf-spectacular**與**drf-yasg**，前者依循OpenAPI 3 Schema來生成，後者則是 Swagger / OpenAPI 2

## OpenAPI

相對於Internal API（Private API），通常是企業內部自行使用的API，而Open API則是聚焦於所有API的使用者，也因此訂定一個不需要閱讀程式碼就能理解API如何使用的規範就變得相當重要，這也是OpenAPI規範（OAS）的宗旨：不論使用哪種程式語言的開發人員，都能在不閱讀程式碼的能了解API的使用方式

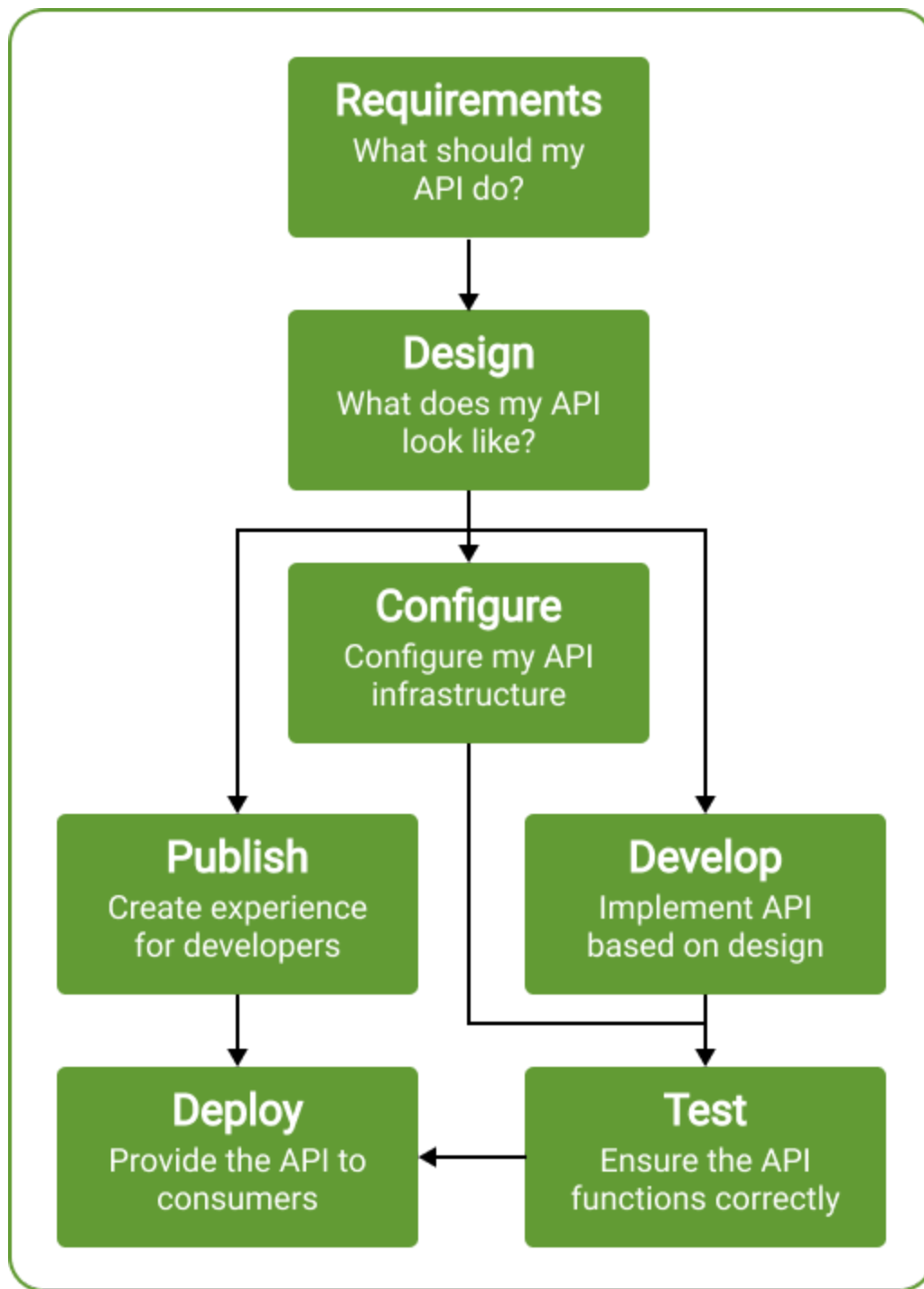
OpenAPI規範具備以下特點：

- 格式：通常以YAML或是JSON格式編寫，便於開發人員或是機器解析

- 描述性：描述了API的端點，請求方法、參數與響應等

在API的生命週期中，OpenAPI規範都能在不同階段幫助到開發人員

- 設計階段：在設計階段OAS可以作為API設計的藍圖，幫助團隊確定API結構
- 開發階段：開發者可以根據OAS建立對應的服務端或是客戶端SDK
- 測試階段：可以根據OAS來生成測試用例
- 維護階段：OAS可以作為API的單一真實來源，方便版本控制
- 棄用階段：OAS可以標記被棄用的API端點



圖源：<https://www.openapis.org/what-is-openapi>

## OpenAPI 2 vs OpenAPI 3

網路上已經有許多資源介紹OpenAPI 2.0與3.0的區別：

- <https://blog.stoplight.io/difference-between-open-v2-v3-v31>
- <https://blog.bitsrc.io/whats-new-in-openapi-3-0-f6604799782>

而我們則根據這些不同來說明**drf-spectacular**與**drf-yasg**之間的區別

- **drf-yasg** :
  - 自動辨認DRF視圖，並且生成對應的文件，但是在處理複雜資料結構的限制可能較多
  - 不支持Open API 3.x的特性，例如 `callback` 與 `links`
    - 表示無法明確定義非同步的操作流程
    - 無法直觀表示API之間的相互依賴關係
- **drf-spectacular** :
  - OpenAPI 3.x引入components概念，因此允許定義可重用的API元素：響應、參數等等
    - 常用元素可以只定義一次，就可以多次引用
    - 確保相同的資料結構在文件中保持一致
  - 支持更詳細的數據模型，例如描述複雜的JSON結構或是深層的嵌套對象

如果是相對簡單的API，可以考慮使用drf-yasg，相對單純。但是如果是大型專案或是更現代的API設計，還是推薦使用drf-spectacular

## drf-spectacular

我們就使用drf-spectacular來自動建立API文件

程式碼：[https://github.com/class83108/drf\\_demo/tree/api\\_doc](https://github.com/class83108/drf_demo/tree/api_doc)

## 基礎使用

- 安裝套件

```
poetry add drf-spectacular
```

- 基礎設定

```
INSTALLED_APPS = [  
    ...  
    "drf_spectacular",  
    ...  
]  
  
REST_FRAMEWORK = {  
    ...  
    "DEFAULT_SCHEMA_CLASS": "drf_spectacular.openapi.AutoSchema"  
}  
  
SPECTACULAR_SETTINGS = {  
    "TITLE": "Note API Documentation",  
    "DESCRIPTION": "API documentation for Note app",  
    "VERSION": "1.0.0",  
    "SERVE_INCLUDE_SCHEMA": False,  
    # OTHER SETTINGS  
}
```

有關SPECTACULAR\_SETTINGS其他額外配置可以參考官方文檔：

<https://drf-spectacular.readthedocs.io/en/latest/settings.html>

- 配置路由

```
from drf_spectacular.views import (
    SpectacularAPIView,
    SpectacularRedocView,
    SpectacularSwaggerView,
)

urlpatterns = [
    ...
    path("api/schema/", SpectacularAPIView.as_view(), name="schema"),
    path(
        "api/schema/swagger-ui/",
        SpectacularSwaggerView.as_view(url_name="schema"),
        name="swagger-ui",
    ),
    path(
        "api/schema/redoc/",
        SpectacularRedocView.as_view(url_name="schema"),
        name="redoc",
    ),
]
```

- **SpectacularAPIView**：會提供一個API的YAML檔

```
openapi: 3.0.3
info:
  title: Your Project API
  version: 1.0.0
  description: Your project description
paths:
  /debug/:
    get:
```

```

    operationId: debug_retrieve
    tags:
    - debug
    responses:
      '200':
        description: No response body
/note/documents/:
get:
  operationId: note_documents_list
  parameters:
    - name: ordering
      required: false
      in: query
      description: Which field to use when ordering the results
      schema:
        type: string
...

```

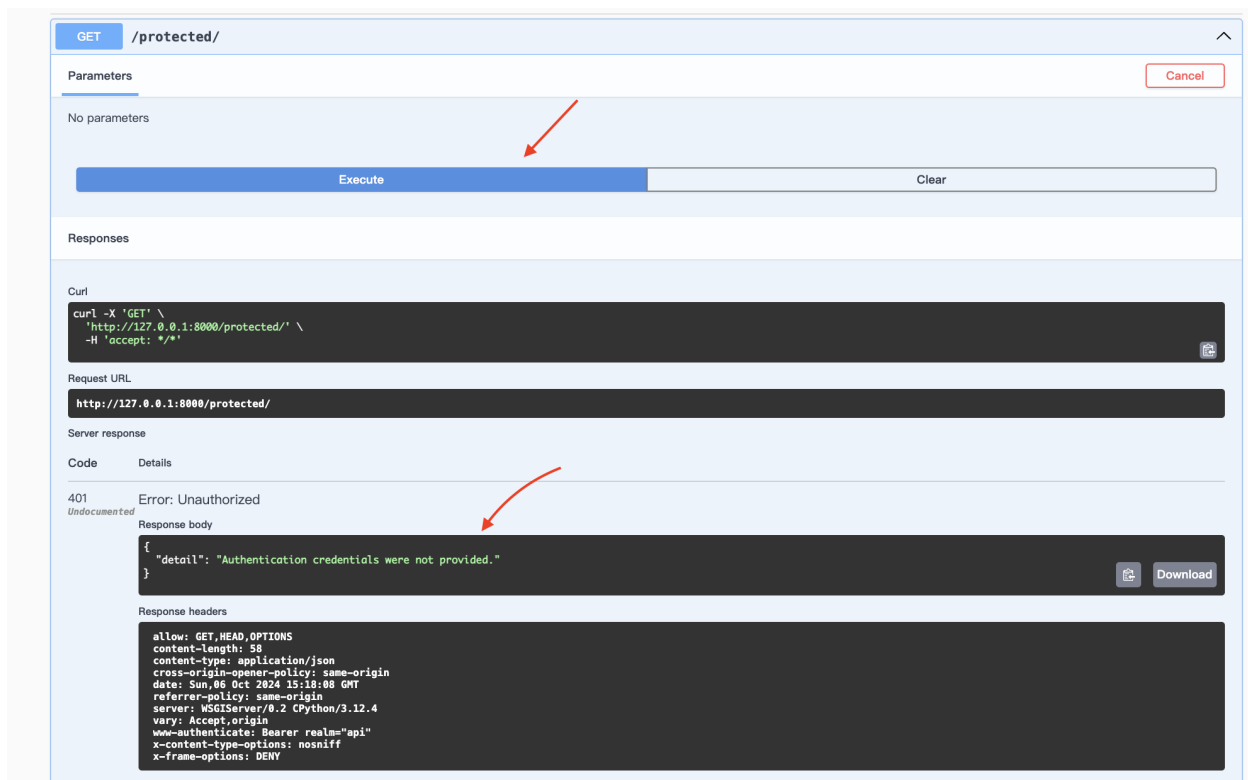
- SpectacularRedocView與SpectacularSwaggerView分別提供不同的UI可以觀看API文件的細節

ReDoc提供清晰易讀的文檔展示方式，不提供測試API的功能

Swagger則是更著重在互動性上，可以讓用戶直接在介面上測試API功能

可以看到我們可以直接測試API，並且看到響應的結果

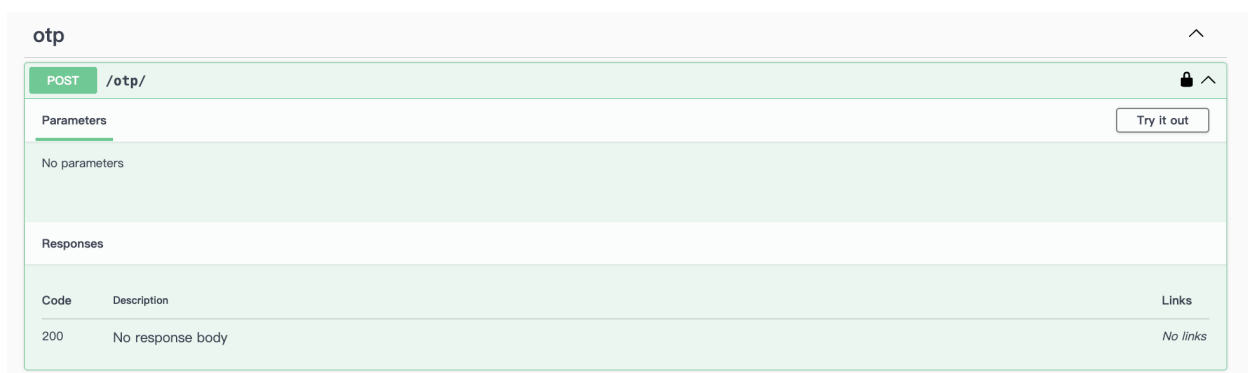




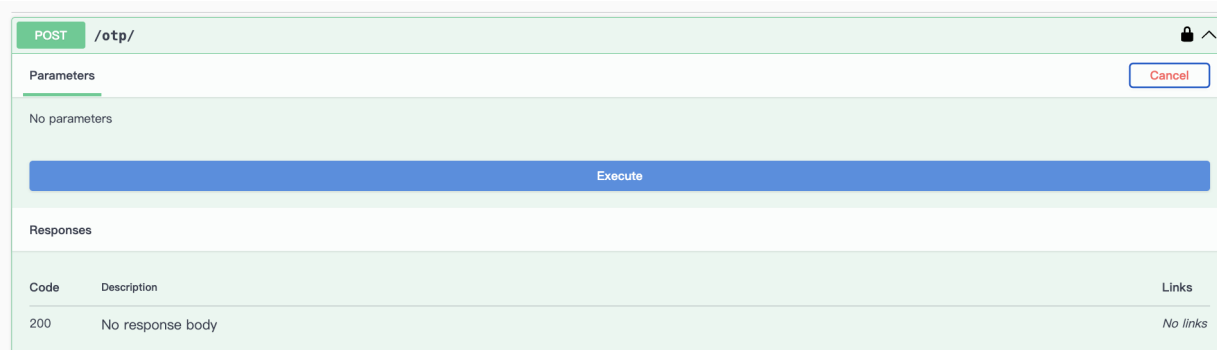
## 客製化API文件

我們來看到我們的otp視圖，在之前**Django REST framework: 掌握 JWT、CORS 和 Cookie 處理技巧**有提到過是用來建立一次性密碼的視圖

但是如果我如果沒有看到程式碼，根本不知道這個API的作用



當我點擊想要測試時，也沒有提供任何schema以及任何形式的表單讓我們知道該怎麼輸入資料以及怎麼輸入資料來完成POST請求



我們會需要使用到 `@extend_schema` 裝飾器來為API自定義Schema，通常會有幾種方式建立出API schema

- 使用類別裝飾器
  - 將整個類別的schema定義好，直接讓整個視圖使用該schema
  - 但是如果不同請求需要定義schema時不夠靈活

```
@extend_schema(  
    ...  
)  
class OTPRequestView(APIView):  
    def post(self, request):  
        # 實現邏輯保持不變  
        ...
```

- 在類別的內部定義schema
  - 可以針對每個方法定義自己的schema
  - 但是如果有多個方法，又要都定義好schema可能會太過冗長

```
class OTPRequestViewInline(APIView):  
    @extend_schema(  
        ...
```

```

    ...
)
def post(self, request):
    # 實現邏輯保持不變
    ...

```

- 使用自動生成的schema
  - 這部分就是我們預設的狀況
  - 優點是非常迅速簡單，但是生成的文檔容易缺乏關鍵資訊
- 單獨使用函式定義schema然後做成裝飾器
  - 可重用性是最高的，可以讓多個視圖共享相同的schema

```

def otp_request_schema():
    return extend_schema(
        ...
    )

@otp_request_schema()
class OTPRequestViewFunction(APIView):
    def post(self, request):

```

這裡因為我的OTP視圖就只有他一個，並且方法也只會有一個POST請求，所以使用類別裝飾器

```

from drf_spectacular.utils import extend_schema, OpenApiResponse

@extend_schema(
    tags=["Authentication"],
    description="Request an OTP for user authentication. The OTP",
    request=OTPRequestSerializer,
    responses={
        status.HTTP_200_OK: OpenApiResponse(

```

```

        response=OTPResponseSerializer, description="OTP ser
    ),
    status.HTTP_400_BAD_REQUEST: OpenApiResponse(
        response=ErrorResponseSerializer, description="Inval
    ),
    status.HTTP_404_NOT_FOUND: OpenApiResponse(
        response=ErrorResponseSerializer, description="User
    ),
},
examples=[
    OpenApiExample(
        "Valid request",
        summary="Request OTP for existing user",
        description='Request an OTP for user "john_doe"',
        value={"username": "john_doe"},
        request_only=True,
    ),
    OpenApiExample(
        "Invalid request",
        summary="Request OTP for non-existing user",
        description="Request an OTP for a user that does not exist",
        value={"username": "non_existing_user"},
        request_only=True,
    ),
    OpenApiExample(
        "Success response",
        summary="OTP sent successfully",
        description="Response when OTP is successfully sent",
        value={"message": "OTP sent successfully."},
        response_only=True,
        status_codes=["200"],
    ),
    OpenApiExample(
        "User not found response",
        summary="User not found error",
        description="Response when the requested user does not exist"
    )
]

```

```

        value={"error": "User not found."},
        response_only=True,
        status_codes=["404"],
    ),
],
)

```

- **tags**：API分類標籤，我們可以將它釘在特定的標籤下，讓同性質的API能夠被集中在一起
- **description**：API端點描述，我們自定義描述該API的功能
- **request**：指定請求體的序列化器
- **response**：定義不同HTTP狀態碼的響應結構
- **examples**：提通不同情境的請求與響應範例

可以看到整個API端點的文檔是不是清晰許多

**Authentication**

**POST /otp/**

Request an OTP for user authentication. The OTP will be sent to the user's registered email address.

**Parameters**

No parameters

**Request body** *required*

application/json

**Examples:**

Request OTP for existing user

Example Value | Schema

```
{
  "username": "john_doe"
}
```

**Example Description**

Request an OTP for user "john\_doe"

**Responses**

Code	Description	Links
200	OTP sent successfully	No links
400	Invalid input data	No links
404	User not found	No links

甚至在response的部分，我們也能透過額外定義序列化器來完善response的schema

```
class OTPResponseSerializer(serializers.Serializer):
    message = serializers.CharField()

class ErrorResponseSerializer(serializers.Serializer):
    error = serializers.CharField()

@extend_schema(
    tags=["Authentication"],
    description="Request an OTP for user authentication. The OTP",
    request=OTPRequestSerializer,
    responses={
        status.HTTP_200_OK: OpenApiResponse(
            response=OTPResponseSerializer, description="OTP ser
        ),
        status.HTTP_400_BAD_REQUEST: OpenApiResponse(
            response=ErrorResponseSerializer, description="Inval
        ),
        status.HTTP_404_NOT_FOUND: OpenApiResponse(
            response=ErrorResponseSerializer, description="User
        ),
    },
    ....
```

Responses		
Code	Description	Links
200	OTP sent successfully	No links
<div>Media type: <input type="text" value="application/json"/></div> <div>Examples: <input type="text" value="OTP sent successfully"/></div> <div>Controls Accept header.</div> <div>Example Value   Schema</div> <div> <pre>{   "message": "OTP sent successfully." }</pre> </div> <div>Example Description</div> <div>Response when OTP is successfully sent</div>		
400	Invalid input data	No links
<div>Media type: <input type="text" value="application/json"/></div> <div>Example Value   Schema</div> <div> <pre>{   "error": "string" }</pre> </div>		
404	User not found	No links
<div>Media type: <input type="text" value="application/json"/></div> <div>Examples: <input type="text" value="User not found error"/></div> <div>Example Value   Schema</div> <div> <pre>{   "error": "User not found." }</pre> </div> <div>Example Description</div> <div>Response when the requested user does not exist</div>		

現在我們重新測試API，看到不論是請求的schema，以及響應的格式，我們都能很清晰的判讀

No parameters

Request body required

Examples:

```
{
  "username": "john_doe"
}
```

Example Description

Request an OTP for user "john\_doe"

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/otp/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -H 'X-CSRFToken: bJG6tn4K1CpKENHIAKbPpqnLOXHjPrIKF6MZKhqAfd4aFjx9edj2444m0mJces' \
  -d '{
    "username": "john_doe"
  }'
```

Request URL

http://127.0.0.1:8000/otp/

Server response

Code	Details
404	Error: Not Found

Response body

```
{
  "error": "User not found."
}
```

Response headers

```
allow: POST,OPTIONS
content-length: 27
content-type: application/json
cross-origin-opener-policy: same-origin
date: Sun, 06 Oct 2024 16:12:57 GMT
referrer-policy: same-origin
server: WSGIServer/0.2 Python/3.12.4
vary: Accept, Cookie, origin
x-content-type-options: nosniff
x-frame-options: DENY
```

Responses

我們此時再次下載YAML檔，可以看到我們新增的描述與schema也一併更新了

```
/otp/:
  post:
    operationId: otp_create
    description: Request an OTP for user authentication. The
      the user's registered email address.
    tags:
      - Authentication
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/OTPRequest'
          examples:
            ValidRequest:
              value:
                username: john_doe
            summary: Request OTP for existing user
            description: Request an OTP for user "john_doe"
```



```

    application/x-www-form-urlencoded:
      schema:
        $ref: '#/components/schemas/OTPRequest'
    multipart/form-data:
      schema:
        $ref: '#/components/schemas/OTPRequest'
    required: true
  security:
    - cookieAuth: []
    - {}
  responses:
    '200':
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/OTPResponse'
          examples:
            SuccessResponse:
              value:
                message: OTP sent successfully.
                summary: OTP sent successfully
                description: Response when OTP is successfully
      description: OTP sent successfully
    '400':
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorResponse'
      description: Invalid input data
    '404':
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/ErrorResponse'
          examples:
            UserNotFoundResponse:



```

```
value:
  error: User not found.
summary: User not found error
description: Response when the requested user
description: User not found
```


我們如果在序列化器定義更多詳細資訊，也能看到schema顯示更清楚的相關資料

```
class OTPRequestSerializer(serializers.Serializer):
    username = serializers.CharField(
        help_text="User's username",
        max_length=150,
        min_length=3,
    )
```

**Authentication**

**POST** /otp/  

Request an OTP for user authentication. The OTP will be sent to the user's registered email address.

**Parameters** 


No parameters

**Request body** *required* application/json

**Examples:**  
Request OTP for existing user

Example Value | Schema

```
OTPrequest {
  username
}
```

 **string**  
maxLength: 150  
minLength: 3  
User's username

**Example Description**  
Request an OTP for user "john\_doe"

## 今日總結

我們在DRF的最後一篇介紹了有關自動生成API文檔的相關知識

- 首先稍微了解到在不同開發階段OpenAPI能夠怎麼幫助到開發流程
- 認識到如果想要套用OpenAPI 3.x的規範，可以使用drf-spectacular快速生成API文件
- 透過 `@extend_schema`，我們能夠針對自動生成時文檔不夠詳細的視圖來修改API文件，使其更完善

## 參考資料

- OpenAPI :
  - <https://www.openapis.org/what-is-openapi>
  - <https://openapi.apifox.cn/>
- drf-spectacular
  - <https://www.rootstrap.com/blog/automating-django-rest-apis-documentation-made-easy-with-drf-spectacular-part-1>
  - <https://www.rootstrap.com/blog/automating-django-rest-apis-documentation-made-easy-with-drf-spectacular-part-2>