

Apps, Views & URL Mappings

إنشاء تطبيق داخل المشروع :

```
1 python manage.py startapp app_name
2
```

تنظيم الملفات في Django

بعد إنشاء المشروع والتطبيقات سوف تظهر لنا عدة ملفات منظمة بالشكل التالي:

أولاً: الملفات داخل مجلد المشروع.

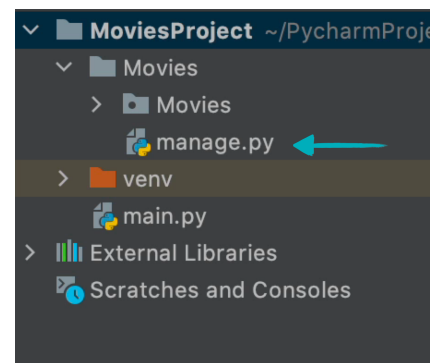
داخل مجلد المشروع Movies

يوجد ملف **manage.py** ويستخدم لإدارة مشروع Django عن طريق عدة أوامر تسمى management commands مثل:

- python **manage.py** runserver
- python **manage.py** startapp
- python **manage.py** migrate

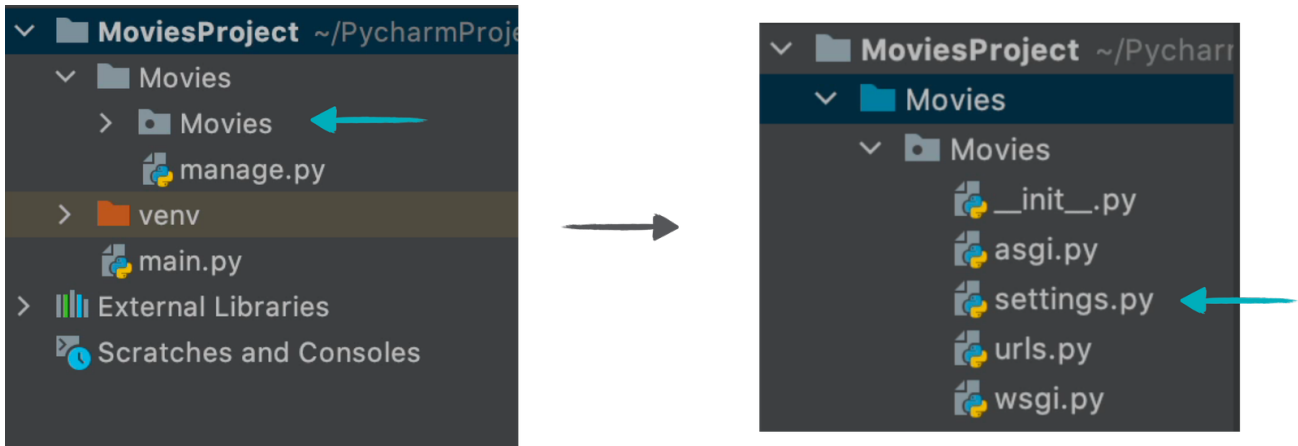
نلاحظ جميع هذه الأوامر تحتوي ملف **manage.py** لذا لابد من وجودنا على نفس مسار هذا الملف عن طريق كتابة `cd Movies`

أيضاً، يوجد Python Package باسم المشروع Movies



ملاحظة: Python Package يحتوي على ملف `__init__.py` والذي يساعدنا على تمييز Package عن Folder.

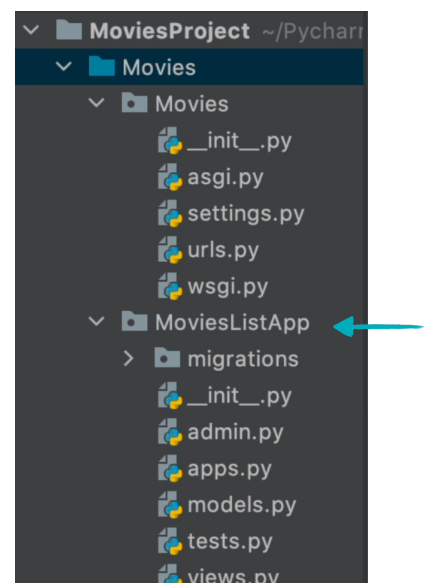
الملفات داخل Python Package:



- ملف `settings.py` و يحتوي على الإعدادات الافتراضية الخاصة بالمشروع والتطبيقات الموجودة في Django.
- ملف `urls.py` وهو global URL mapping خاص لعمل URL mapping سواء لصفحات views أو child urls.
- ملف `asgi.py` يحتوي الإعدادات لتشغيل المشروع على **ASGI (Asynchronous Server Gateway Interface)**.
- ملف `wsgi.py` يحتوي الإعدادات لتشغيل المشروع على **WSGI (Web Server Gateway Interface)**.

ثانياً: الملفات داخل مجلد التطبيق.

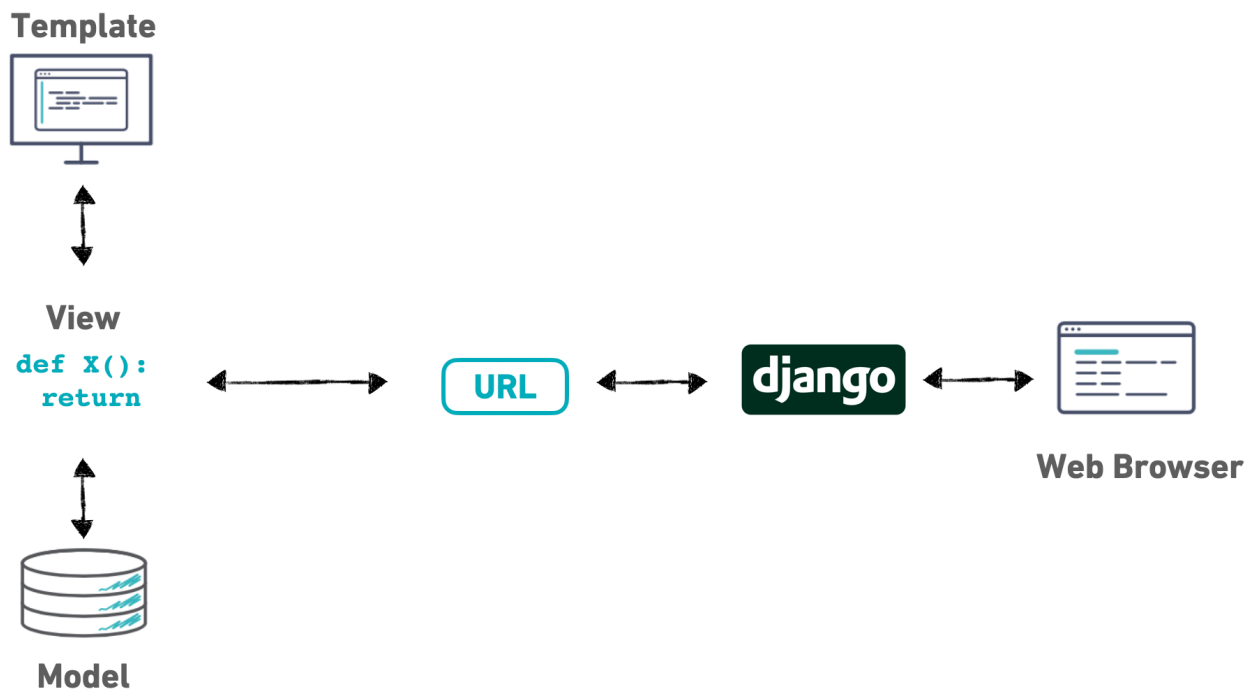
- مجلد `migrations` وبداخله نقوم بحفظ التعديلات التي تمت على قاعدة البيانات.
- ملف `admin.py` خاص بموقع `admin` والذي من خلاله يستطيع `admin` التعديل على البيانات عن طريق `Interface` جاهزة من Django.
- ملف `apps.py` يحتوي على `metadata` والإعدادات الخاصة بالتطبيق.
- ملف `models.py` نقوم بكتابة `model` بداخله لنتمكن من التعامل مع البيانات.
- ملف `views.py` نقوم بكتابة `Django views` بداخله وهي تعبر عن المنطق الخاص بالتطبيق حيث تستقبل `Http Request` ثم تقوم بمعالجته وإرجاع `Http Response`.
- ملف `tests.py` ونستخدمه لاختبار الأكواد البرمجية والتأكد من



أنها قد كتبت بشكل صحيح ويمكن كتابة أنواع مختلفة من tests مثل: unit و functional integration.

نظرة عامة على آلية عمل المشروع في Django

لتوضيح كيف تعمل أجزاء MVT بداخل مشروع Django، انظر للمثال التالي:



- لنفترض أن لدينا مستخدم قام بالدخول على رابط موقع إحدى المكتبات لاستعراض قائمة الكتب الموجودة، هذا المستخدم سوف يمر بالخطوات التالية:
- يرسل المتصفح http request إلى Django Server.
 - يقوم Django بمطابقة request مع ملف url وبالتالي سوف يحدد view المسؤولة عن إرجاع قائمة الكتب ويرسل لها http request.
 - تقوم هذه view باستقبال http request ومعالجته عن طريق إرجاع البيانات من Model مثلا: أسماء الكتب المخزنة في قاعدة البيانات وصورها.
 - بعد ذلك سوف تقوم view بالتواصل مع جزء Template ثم إرجاع http response وبالعالم سوف يكون على شكل صفحة html يتم عرضها للمستخدم (صفحة تحتوي معلومات الكتب).

مفهوم HttpRequest و HttpResponse

أولاً: مفهوم HttpRequest

عند زيارتنا لأي صفحة Web على سبيل المثال (<https://www.example.com/page>) يقوم المتصفح بإنشاء HttpRequest يتم إرساله إلى Web Server و يتكون HttpRequest من ٤ أجزاء وهي:

- الأول: Method
- الثاني: Path
- الثالث: Headers
- الرابع: Body

ويمكن توضيح هذه الأجزاء في المثال التالي:

Method Path

GET /page HTTP/1.1

Headers

Host: www.example.com

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:15.0) Firefox/15.0.1

Cookie: sessid=abc123def456

أولاً: أنواع HTTP Methods

يوجد هناك أنواع مختلفة من HTTP Methods تمثل عمليات (CRUD) على البيانات، ومن أشهر هذه الأنواع:

1. نوع GET يستخدم لاسترجاع البيانات (Read) من remote page.
2. نوع POST يستخدم لإضافة البيانات (Create) إلى remote page.
3. نوع PUT يستخدم لتحديث البيانات (Update) في remote page.
4. نوع DELETE يستخدم لحذف البيانات (Delete) من remote page.

عند كتابتنا Web Application نستخدم في الغالب GET، وعند التعامل مع Forms نستخدم POST، وأخيراً عند إنشاء REST APIs سوف نحتاج لاستخدام PUT و DELETE.

ثانياً: المسار أو Path الخاص بصفحة Web.

ثالثاً: Headers

يحتوي معلومات إضافية (metadata) عن HttpRequest وهي كالآتي:

- المضيف (Host) يُمكن web server من معرفة ما هو الموقع الذي يجب إرجاعه وعرضه.
- (User-Agent) يقوم المتصفح بإرسال نوع نظام التشغيل ونسخته وبالتالي عرض صفحات

تتوافق مع نوع الجهاز.

- Cookie يُمثل أجزاء صغيرة من المعلومات يقوم الموقع بتخزينها في المتصفح تساعد في التعرف على المستخدم عند زيارته للموقع في المرة القادمة.

رابعًا: Body

نلاحظ أن المتصفح لم يقوم بإرسال جزء خاص Body في المثال السابق وذلك لأننا فقط نقوم بزيارة للصفحة ولكن لو كنا نقوم بدخول صفحة Login مثلاً فإننا سوف نحتاج لإرسال معلومات وبالتالي سوف نقوم بكتابة جزء Body بحيث يحتوي معلومات مثل: (Username, Password).

ثانيًا: مفهوم HttpResponse

يتكون HttpResponse من ٣ أجزاء وهي:

- الأول: Status
- الثاني: Headers
- الثالث: Body

ويمكن توضيح هذه الأجزاء في المثال التالي:

Status
HTTP / 1.1 200 OK
Headers

Server: nginx
Content-Length: 18132
Content-Type: text/html
Set-Cookie: sessid=abc123def46

<!DOCTYPE html><html><head>... Body

أولًا: أنواع Status Code

تمثل Status Code بقيمة عددية ثم نص يصف معنى Code مثل: (OK 200) كما في المثال بالأعلى وتعني أن request تم بنجاح، يمكن تقسيم Status Code لمجموعات كالتالي:

1. 100-199 مجموعة الأرقام هذه تمثل protocol changes أو الحاجة للمزيد من البيانات.
2. 200-299 مجموعة الأرقام هذه تعني أنه تمت معالجة Response بنجاح وأشهرها (OK 200).
3. 300-399 مجموعة الأرقام هذه تعني أنه قد تم نقل الصفحة المطلوبة لعنوان آخر وأشهرها (302 Found) و (301 Moved Permanently).
4. 400-499 مجموعة الأرقام هذه تعني أنه لم تتم معالجة request بسبب مشكلة في client-side مثل (401 Unauthorized) و (403 Forbidden) و (404 Not Found).
5. 500-599 مجموعة الأرقام هذه تعني أنه لم تتم معالجة request بسبب مشكلة في server-side مثل (500 Bad Request) و (502 Bad Gateway) و (503 Service Unavailable) و (504 Gateway Timeout) و (505 HTTP Version Not Supported) و (506 Variant Not Negotiable) و (507 Insufficient Storage) و (508 Precondition Failed) و (509 Bandwidth Limit Exceeded) و (510 Not Extended) و (511 Network Authentication Required).

ثانيًا: Headers


يحتوي معلومات إضافية وهي كالآتي:

- Server: يقوم بعكس ما يقوم به User-Agent حيث أن Server يقوم بإخبار Client بنوع Software المستخدم.
- Content-Length: يمثل كمية البيانات التي يجب قرائتها من Server للحصول على Body.
- Content-Type: يمثل نوع البيانات المرسله حيث تساعد Client باختيار طريقة عرض كل نوع من البيانات.
- Set-Cookie: تستخدم لتعيين cookie على المتصفح.

ثالثًا: Body

يمثل ملف HTML الذي تم الحصول عليه وفي المثال بالأعلى تم عرض جزء بسيط من هذا الملف.


[معلومات إضافية]: للاطلاع على مزيد من أنواع Status Code، قم بالدخول على الموقع:



HTTP response status codes - HTTP | MDN

HTTP response status codes indicate whether a specific HTTP request has been successfully completed. Responses are grouped i...

developer.mozilla.org

 HTTP response status codes - HTTP | MDN • developer.mozilla.org

إنشاء View

يعتبر View أحد أهم أجزاء MVT حيث يقوم بتنظيم التفاعل بين Model و Template و يعبر عن المنطق الخاص بالتطبيق حيث يقوم باستقبال Http Request ثم معالجته وإرجاع Http Response.

أنواع Views

تقدم Django نوعين من views وهي:

النوع الأول: function-based views وهنا نعبر عن view عن طريق كتابة python function.

النوع الثاني: class-based views وهنا نعبر عن view عن طريق كتابة python class ويمكن

تطبيق جميع المفاهيم عليه مثل: class inheritance وغيره.

إنشاء Function-Based View

لإنشاء View قم باتباع الخطوات التالية:

- فتح ملف `views.py` الموجود بداخل مجلد التطبيق `MoviesListApp` وكتابة التالي.

```
1 from django.http import HttpResponse
2 def Movie_Name(request):
3     return HttpResponse("This page displays information about th
e movies")
```

حتى نستطيع عرض الصفحة التي قمنا بكتابتها لابد من عمل URL Mapping.

URL Mappings

تعلمنا سابقا طريقة اضافة المسارات للمشروع ، و استخدمنا الملف الأساسي `urls.py` داخل مجلد المشروع الأساسي لإضافة مسارات التطبيق الذي انشئناه ، و هذا تعتبر اضافة `global` او عامة .

من اجل ترتيب المشروع بشكل افضل ، و من اجل الاستفادة ايضا من توحيد مسار التطبيق الذي نعمل عليه ، سنقوم بإنشاء ملف `urls.py` خاص بكل تطبيق . و من ثم استيراد المسارات المضافة في ذلك الملف الى ملف المشروع الأساسي . بحيث يصبح كل تطبيق مسؤول عن المسارات الخاصة به في ملف منفصل ، مما يساعدنا على ترتيب الكود بشكل افضل .

بداية انشاء ملف جديدة تحت مسمى `urls.py` في مجلد التطبيق الذي تريد العمل عليه

داخل ملف ال `urls.py`

نقوم بالبداية بإستيراد ال `path` من `django.url` كالتالي :

```
from django.urls import path
```

بعدها نستورد الموديول `views` الخاص بالتطبيق :

```
from . import views
```

نقوم بإضافة الـ `app_name` و الذي سيستخدمه الـ `django` كمعرف للتطبيق `namespace` :

```
app_name = "my_first_app"
```

لاحقا ، ننشئ متغير `urlpatterns` و بداخله نضيف قائمة المسارات التي سيتعامل معها هذا التطبيق .

لاحظ هنا أننا أيضا أضفنا للـ `path` باراميتير جديد هو `name` ، و هذا مُعرف للمسار يمكن الاستفادة منه لاحقا لجلب رابط المسار ضمن المشروع .

```
1 urlpatterns = [  
2     path("path/", views.hello, name="path")  
3 ]
```

إضافة مسارات التطبيق للمشروع

اخيرا ، نقوم بإضافة المسارات الموجودة في التطبيق الى المسار الاساسي للمشروع و ذلك من خلال استخدام `include` .

في ملف `urls.py` الخاص بالمشروع (ليس التطبيق)، نستودر `include` و من ثم نقوم بإستخدامها لإضافة مسارات التطبيق ، بحيث نعطيه اسمه الباكج نقطة اسم الملف الذي يحتوي على المسارات ، كالتالي :

```
1 urlpatterns = [  
2     path('admin/', admin.site.urls),  
3     path("movies/", include("my_app.urls"))  
4 ]
```

بالإمكان الآن ، استخدام مسار مبدئي ، يتم استخدامه بشكل تلقائي عند استدعاء اي مسار من التطبيق . و ذلك من خلال توفير قيمة المسار المبدئي في المسار الخاص بتطبيقك (الذي استخدمنا معه الـ `include`) .