

مقدمة عن تطوير الويب باستخدام Django

مقدمة عن Django

يعتبر Django من أشهر Framework مفتوحة المصدر الخاصة بتطبيقات الويب ويسخدم في مواقع مشهورة مثل: Disqus, Instagram, Pinterest.



ماهي Web Frameworks؟



يوجد لدينا Web Frameworks مشهورة مثل: Django، Vue، Angular، ReactJS ويمكن تصنيف هذه Frameworks إلى ثلاثة أنواع كالتالي:

- **أولاً: Front-end Frameworks** وهي تعبر عن كل ما يراه المستخدم أو الأجزاء التي يتعامل معها المستخدم أثناء تصفح واجهات الموقع مثل الألوان والخطوط و dropdown menus وبشكل عام مكونات HTML و CSS و JavaScript.
- **ثانياً: Back-end Frameworks** وهي ما يمكن Front-end Frameworks من العمل ويقوم بحفظ البيانات الخاصة بالموقع وتشمل أيضا server و application.
- **ثالثاً: Full-stack Frameworks** وهي تشمل النوعين السابقين، حيث يعمل Full-stack Developer على Client-side و Server-side .

Front-end

1. How things look
2. Images, content, structure
3. HTML, CSS, JavaScript

Back-end

1. How things work
2. Logic & data
3. Ruby, Python, PHP, Java, etc

أما **Django** فهو **Full-stack Framework** بحيث يكون مسؤول عن بناء واجهات صفحات الويب عن طريق **Templates** ويتفاعل مع قواعد البيانات عن طريق **Models**. ويمكن أيضا أن نستخدم Django ك **Back-end Framework** ونحسن واجهات Django أو نضيف لها **Interactivity** عن طريق **Front-end Frameworks** مثل: **ReactJS**.

مميزات Django Framework:

- يعتبر **batteries included web framework** والمقصود هو أن Django تقدم العديد من المميزات مثل: **libraries** و **packages** التي تسهل عملية تطوير المواقع على المبرمج ومن أمثلة ذلك: **ORM** التي تساعد في بناء قواعد البيانات باستخدام لغة **python** دون الحاجة لكتابة أكواد بلغة **SQL**، أيضا توفر واجهات **Admin** جاهزة وغيرها.
- يساعد في إنشاء تطبيقات تتميز بأنها مرنة (**reliable**) بحيث يكون لديها القدرة في أنها تكبر لتتناسب مع احتياجاتنا.
- يساعد في إنشاء تطبيقات آمنة (**secure**) وسهلة التعلم.
- لديها **built-in database querying** و **URL mapping** و **template rendering**.

المهارات التي نحتاجها لتعلم Django هي:

- معرفة أساسيات لغة **Python**
- معرفة أساسيات **HTML, CSS, JavaScript**
- معرفة أساسيات قواعد البيانات **SQL**



الأدوات التي نحتاجها لتعلم Django هي:

- تحميل **Python** نسخة 3 وأعلى.
- تحميل **Editor** مناسب مثل **PyCharm**.
- تحميل **Django** نسخة 3 وأعلى.

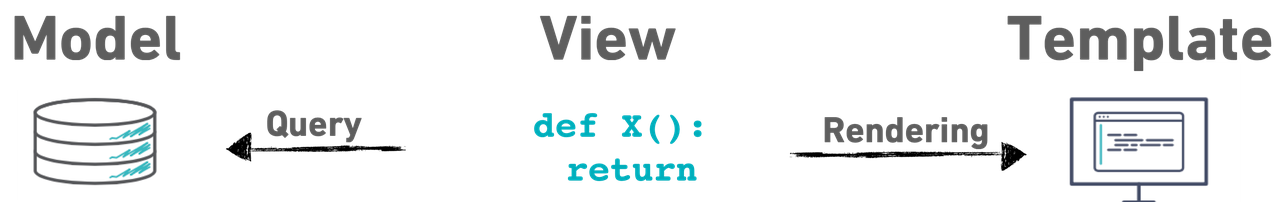


مفهوم MVC و MVT

كلمة **MVC** هي اختصار (**Model View Controller**) وهو يمثل Architectural Pattern ويستخدم كوسيلة لفصل واجهة المستخدم (User Interface) للتطبيق إلى ثلاثة جوانب رئيسية:
أولاً: **Model**: يعبر عن البيانات التي يتم عرضها في الواجهات التي تظهر للمستخدم بالإضافة إلى قواعد تغيير البيانات ومعالجتها.
ثانياً: **View**: يعبر عن الواجهات التي يتم عرضها للمستخدم.
ثالثاً: **Controller**: يعبر عن الجزء الذي ينظم التفاعل بين Model و View ويتعامل مع المستخدم.



في Django نستخدم نموذج مشابه يسمى **MVT** وهو اختصار (**Model View Template**) حيث يمثل كل جزء التالي:
أولاً: **Model**: يعبر عن البيانات التي يتم عرضها في الواجهات التي تظهر للمستخدم بالإضافة إلى قواعد تغيير البيانات ومعالجتها.
ثانياً: **View**: يعبر عن الجزء الذي ينظم التفاعل بين Model و Template.
ثالثاً: **Template**: يعبر عن الواجهات التي يتم عرضها للمستخدم.



في Django عادة ما يتم تمثيل View على شكل Function أو Class يقوم بعمل Query على بيانات Model ثم يقوم بعرضها على الواجهات Template.

مثال:

- لنفرض أنك تقوم باستعراض قائمة الكتب من موقع إحدى المكتبات، في هذه الحالة يمثل تمثيل MVT كالتالي:
- الواجهة أو الصفحات التي تحتوي على قائمة الكتب (تمثل جزء Template).
- البيانات المخزنة في قواعد البيانات الخاصة بالكتب مثل عنوان وصورة الكتاب (تمثل جزء Model)

- أخيرا (جزء View) ينظم التفاعل بين Model و Template و يمكن أن يتم تمثيل View على شكل (BookList Function) هذه Function تقوم بإرجاع البيانات الخاصة بالكتب عن طريق عمل Query على قاعدة البيانات، ثم تقوم بعرض هذه البيانات (الكتب) على الواجهات.

الفرق بين MVC و MVT

- في نموذج MVC يقوم Controller بتنظيم التفاعل بين Model و View.
- في نموذج MVT يقوم جزء View بتنظيم التفاعل بين Model و Template.
- في نموذج MVC يمثل View جزء الواجهات.
- في نموذج MVT يمثل Template جزء الواجهات.
- في نموذج MVC نقوم ببرمجة الثلاث مكونات بنفس لغة البرمجة.
- في نموذج MVT يتم عمل Template بلغة مختلفة، على سبيل المثال: في Django نقوم بكتابة Model و View بلغة Python أما Template نقوم بكتابتها عن طريق HTML.

تحميل Django Framework

تحميل Django Framework لنظام MacOS

لتحميل الأدوات التي نحتاجها لتعلم Django على نظام MacOS، قم باتباع التعليمات في الفيديو التالي:

<https://satr.codes/courses/ba82119d-6d04-4ca1-bd4d-9667b04525b6/session/bc0db2e9-949b-45ca-a46c-5fba3390899f/view>

 منصة سطر التعليمية • satr.codes

تحميل Django Framework لنظام Windows

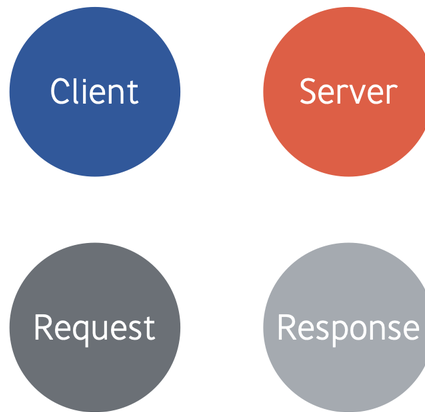
لتحميل الأدوات التي نحتاجها لتعلم Django على نظام Windows، قم باتباع التعليمات في الفيديو التالي:

<https://satr.codes/courses/ba82119d-6d04-4ca1-bd4d-9667b04525b6/session/403b3199-8d72-43eb-8796-94ac4ab8e692/view>

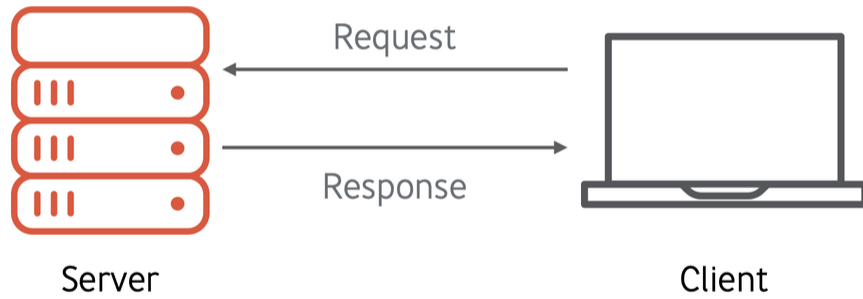
 منصة سطر التعليمية • satr.codes

مفهوم World Wide Web

قبل البدء في تطوير مواقع الويب، نحتاج أولاً إلى معرفة مفهوم الويب بشكل عام. الويب هو عبارة عن نظام من صفحات الويب التي نستطيع الوصول إليها من خلال الإنترنت. مما يعني أن الويب ليس الإنترنت، بل هو واحد من التطبيقات التي تم إنشاؤها على الإنترنت. هناك بعض الأجزاء في World Wide Web نحتاج إلى فهمها ومعرفة الغرض من وجودها لكثرة استخدام مصطلحاتها أثناء تطوير تطبيقات الويب وهي Client, Server, Request, Response.

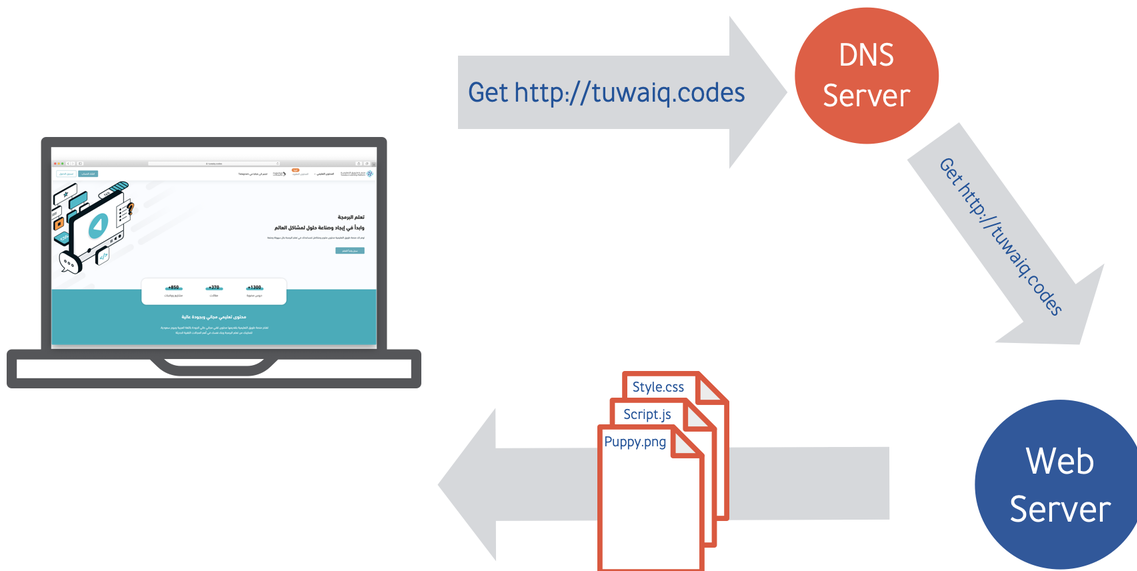


عندما يتصفح المستخدم الإنترنت من خلال PC, Laptop, Smart Phone, iPad في هذه الحالة يعتبر **مستخدم (Client)**، بحيث أنه عندما يبحث عن موقع معين من خلال وضع رابط هذا الموقع في المتصفح فهو في هذه الحالة يقوم بإرسال **طلب (Request)** أي طلب لهذا الموقع، في هذه الأثناء يقوم **الخادم (Server)** باستقبال طلب المستخدم للموقع ثم يقوم بعرض هذا الموقع للمستخدم **كرد (Response)** لطلب المستخدم.



مثال:

عندما تقوم بفتح المتصفح (safari أو google chrome مثلاً) وتقوم بالبحث عن منصة طويق التعليمية من خلال الرابط `www.tuwaiq.codes` في هذه الحالة أنت Client، ويقوم بعدها DNS باستقبال طلبك لتحويل ذلك الرابط إلى IP address الخاص بمنصة طويق التعليمية، حتى يرسل طلبك إلى Server منصة طويق التعليمية، بعد ذلك يقوم Server منصة طويق التعليمية بالرد على طلبك من خلال عرض الموقع لك.



ما هو DNS؟ هو اختصار لمصطلح *Domain Name System* يستخدم لربط اسم الموقع أو رابط الموقع بعنوان *IP address* لذلك الموقع، لأنه من الصعب أن نقوم بحفظ *IP address* لكل المواقع التي نريد زيارتها، فيقوم *DNS* بالمساعدة بحيث يربط اسم كل موقع بالعنوان الحقيقي للخادم *Server* الخاص به. فتبحث أنت باستخدام اسم الموقع مثل *google.com* ويقوم *DNS* بإيجاد العنوان المقابل *216.58.212.110* وإرسال *Request* للخادم *Server* الخاص بذلك الموقع.

Domain Name	IP Address
google.com	216.58.212.110
tuwaiq.codes	46.101.121.244

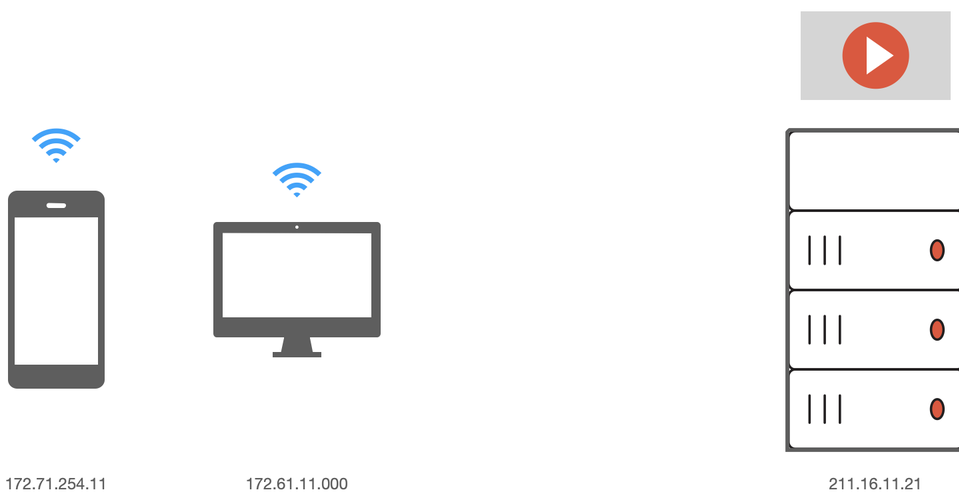
العناوين ليست دقيقة، مجرد مثال .

كيف يعمل Internet

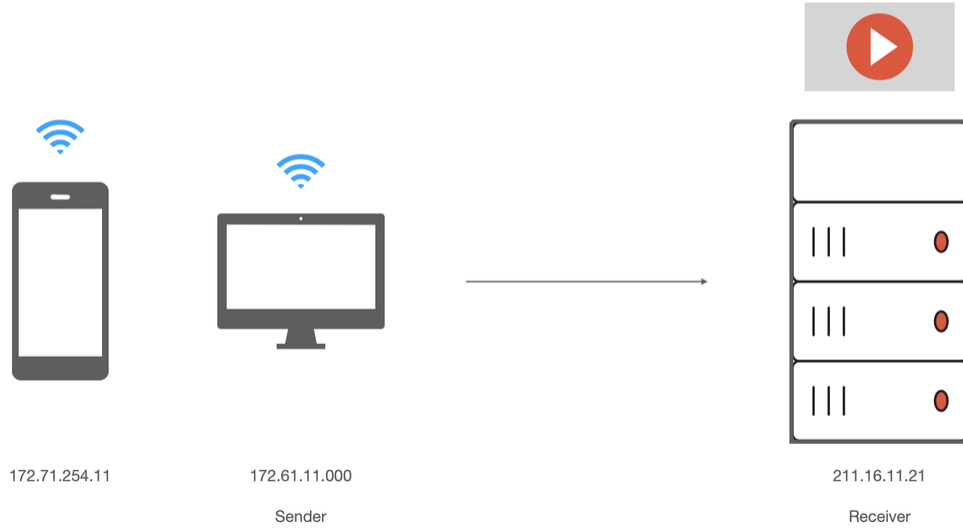
لنفترض أنك تريد الوصول لبيانات معينة من خلال Laptop أو Smart Phone موصول بشبكة الانترنت من خلال wifi .
والبيانات التي تريد الوصول لها مثلاً عبارة عن مقطع Youtube مخزن في مركز بيانات شركة Youtube . لكن كيف تستطيع الوصول لذلك المقطع؟



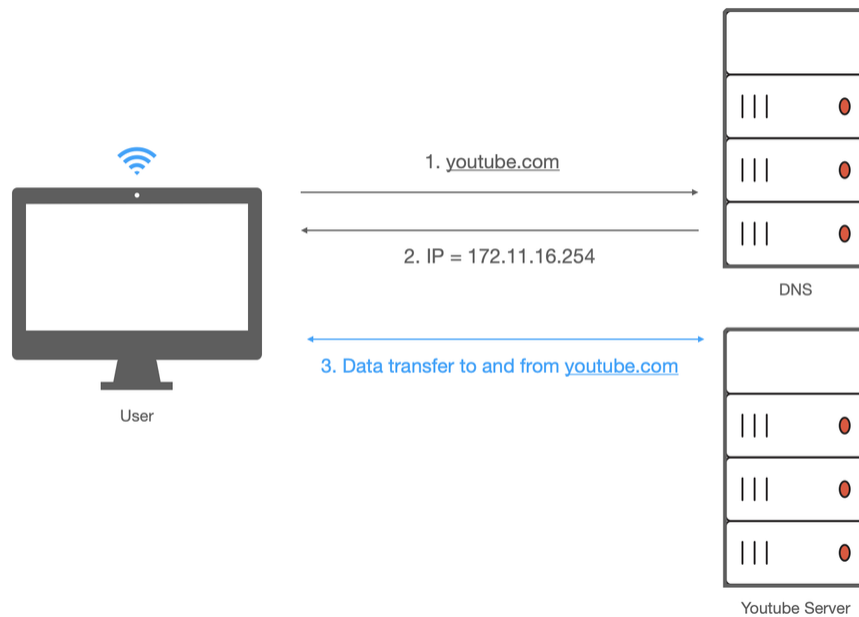
أي جهاز موصول بشبكة الانترنت يكون معرف من خلال مجموعة من الأرقام تسمى IP address. تلك العناوين يتم توفيرها من خلال *ISP (Internet Service Provider)* والتي تقوم بتوفير عنوان مختلف لكل جهاز في شبكة الانترنت. كذلك Server الذي يحتوي على المقطع الذي تريد مشاهدته يكون له عنوان IP خاص به.



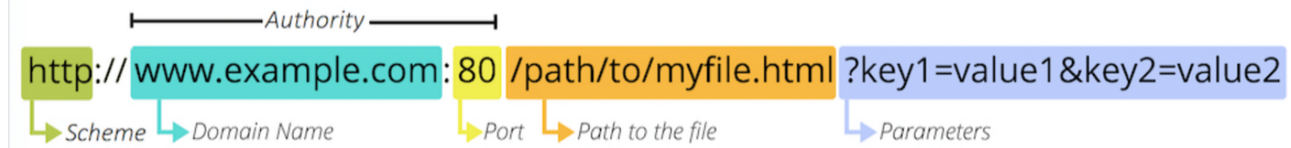
فمن خلال عناوين IP يكون هناك مُرسل Sender ومُستقبل Receiver.



وهذا يعني أن موقع Youtube يملك عنوان IP خاص به، لكن للبحث عنه والوصل إليه لا يشترط أن نقوم بحفظ عنوان IP. يمكننا استخدام الرابط `youtube.com` وذلك بمساعدة *DNS* الذي يقوم بربط كل عنوان IP باسم يسهل حفظه والبحث عنه باستمرار.

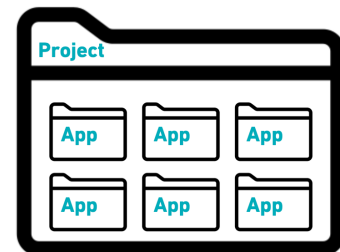


مكونات URL



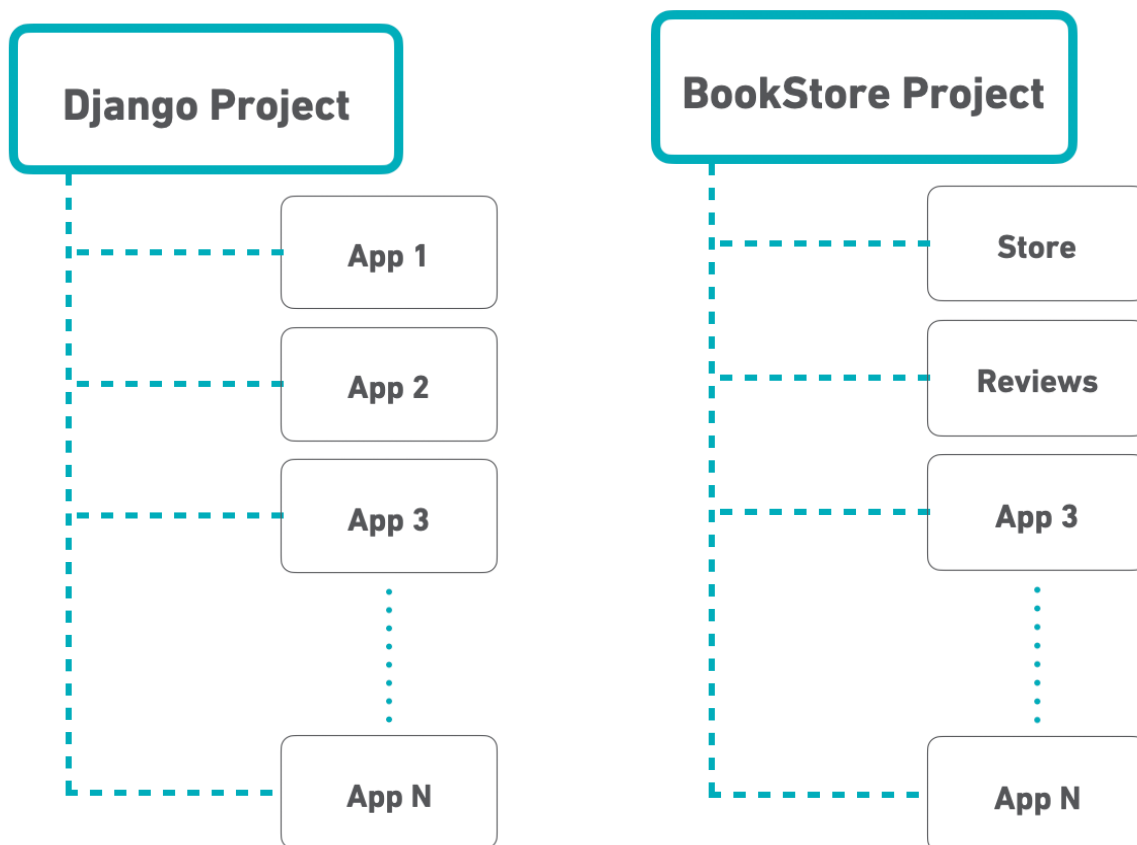
مفهوم المشروع والتطبيق في Django

يمثل المشروع والتطبيق مفهومين مختلفين في Django حيث أن:
 المشروع (project): يشير للخطوة التي نقوم فيها بتحميل Django مع الإعدادات الخاصة به.
 التطبيق (application): يمثل مجموعة من models و views و templates و URLs.
 المشروع يمثل الموقع (website) والذي يتكون من عدة تطبيقات (applications) مثل: blog و forum ويمكن إعادة استخدام التطبيقات في مشاريع أخرى.



مثال:

في Django نقسم المشروع لعدة تطبيقات على حسب الوظائف الخاصة فيها، لو كان لدينا مشروع bookstore يحتوي عدة تطبيقات مثل store app ، reviews app وغيره، نلاحظ أن كل تطبيق يتميز عن الآخر بالوظائف الخاصة التي يقدمها بحيث يمكن استخدام أحد هذه التطبيقات في مشروع آخر (مثال: استخدام تطبيق reviews في مشروع Movies).



إنشاء مشروع في Django

لإنشاء مشروع جديد في Django نقوم باتتباع الخطوات التالية:

1. فتح برنامج PyCharm.
2. اختيار New Project.
3. تسمية المشروع MoviesProject. نلاحظ أن PyCharm أنشأ بيئة افتراضية (Virtual Environment) للمشروع بشكل تلقائي وفائدة وجود بيئة افتراضية هي لحفظ ملفات Python packages للمشروع بشكل منفصل عن بقية (System Packages).
4. التأكد من اختيار أحدث نسخة Python.
5. فتح PyCharm Terminal وكتابة الأوامر التالية.

تحميل django بداخل Virtual Environment.

```
python -m pip install django
```

ملاحظة: أثناء تحميل Django قد يظهر لك التحذير التالي.

```
WARNING: You are using pip version xx.x.x; however, version x
x.x.x is available.
```

عندها يمكن تحديث pip عن طريق كتابة:

```
python -m pip install --upgrade pip
```

إنشاء مشروع جديد بإسم Movies.

```
django-admin startproject Movies
```

سوف نلاحظ أن هذا الأمر أنشأ Python package بإسم المشروع Movies وملف manage.py

ملاحظة: إذا كانت الملفات غير ظاهرة لك قم بالضغط على الزر الأيمن واختيار Reload from the disk بعد ذلك سوف تظهر لنا الملفات.

التأكد من أن المشروع يشتغل بشكل صحيح على server.

```
- cd Movies
- python manage.py runserver
```

الآن، سوف يظهر لنا رابط (<http://127.0.0.1:8000> localhost) نقوم بالدخول عليه، ونلاحظ ظهور العبارة the install worked successfully, congratulation وهذا يعني أن المشروع يشتغل بشكل صحيح على server.

يمكن إيقاف server عن طريق الضغط على (Ctrl+C).

ملاحظة: أثناء تشغيل server قد يظهر لك التحذير التالي.

```
You have 18 unapplied migration(s). Your project may not work
properly until you apply the migrations for app(s): admin, aut
h, contenttypes, sessions.
```

```
Run 'python manage.py migrate' to apply them.
```

وهذا بسبب أن Django تنشأ تطبيقات افتراضية عند إنشاء المشروع مثل: تطبيق Admin و تطبيق auth و sessions و contenttypes ولأن هذه التطبيقات تم إنشائها من دون عمل migration يظهر لنا هذا التحذير لذلك يمكننا تجاهل التحذير أو تطبيق migration عن طريق الأمر.

```
python manage.py migrate
```

وسنتعرف على هذا الأمر بشكل مفصل عند العمل مع قواعد البيانات.

ملاحظة: يمكنك استعراض التطبيقات الافتراضية بداخل ملف settings.py ثم installed apps.

```

32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'MoviesListApp'
41 ]

```

إنشاء تطبيق في Django

لإنشاء تطبيق جديد في Django نقوم باتتباع الخطوات التالية:

1. نتأكد بأننا موجودين بداخل مجلد المشروع (Movies) عن طريق كتابة `cd Movies`.
2. نقوم بكتابة الأمر الخاص بإنشاء التطبيق.

```
python manage.py startapp MoviesListApp
```

في السطر السابق قمنا بتسمية التطبيق `MoviesListApp`.

بعد كتابة الأمر السابق سوف نلاحظ إنشاء مجلد للتطبيق ويحتوي على ملفات `models` و `views` و `templates` و `URLs`.

3. فتح ملف `settings.py` الموجود بداخل مجلد `Movies`

4. تعديل `INSTALLED_APPS` list وإضافة التطبيق `MoviesListApp` حتى نتمكن من Django من الوصول للملفات والمجلدات الخاصة بكل تطبيق (على سبيل المثال: مجلد `templates` الخاص بالتطبيق).

يمكن إضافة اسم التطبيق بطريقتين:

الطريقة الأولى:

```

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'MoviesListApp'
]

```

الطريقة الثانية:

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'MoviesListApp.apps.MovieslistappConfig'  
]
```

ملاحظة: نستطيع إيجاد MovieslistappConfig بداخل ملف apps.py

تذكر أن الأمر المستخدم لإنشاء التطبيق هو:

`python manage.py startapp AppName`

بينما الأمر الخاص بإنشاء المشروع هو:

`django-admin startproject ProjectName`

سؤال:

قم بالدخول للرابط التالي (<http://127.0.0.1:8000/admin>)، هل يمكنك تفسير وجود هذه الصفحة دون إنشائها؟

- وجود تطبيق خاص بالأدمن في التطبيقات
- وفي url

تنظيم الملفات في Django

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

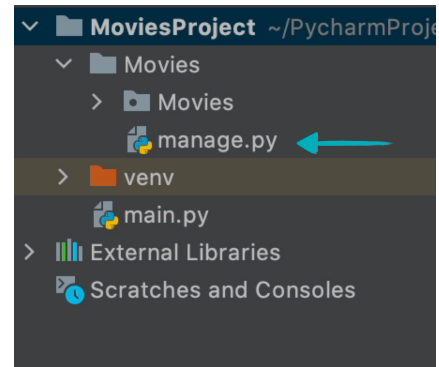
أولاً: الملفات داخل مجلد المشروع.

داخل مجلد المشروع Movies

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

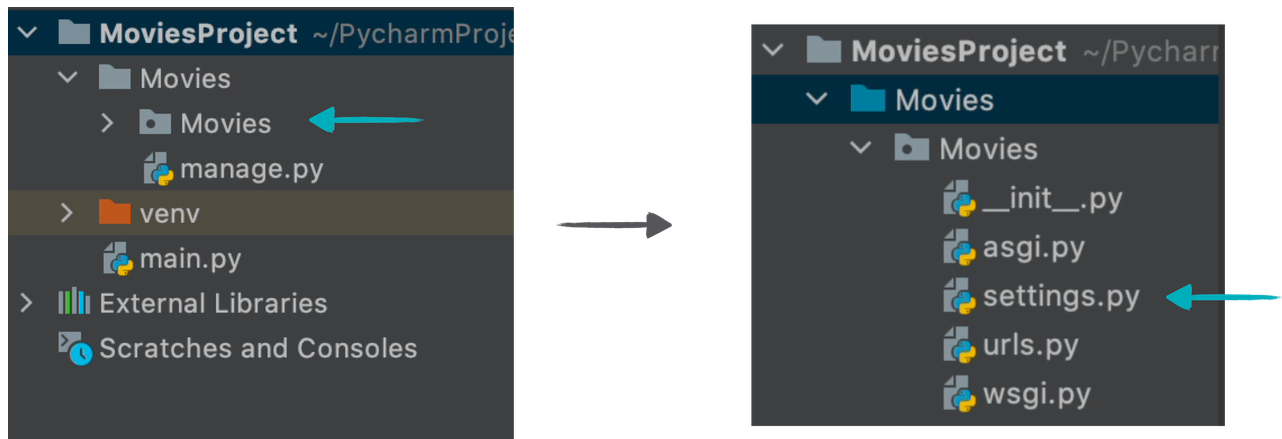
- python **manage.py** runserver
- python **manage.py** startapp
- python **manage.py** dbshell

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

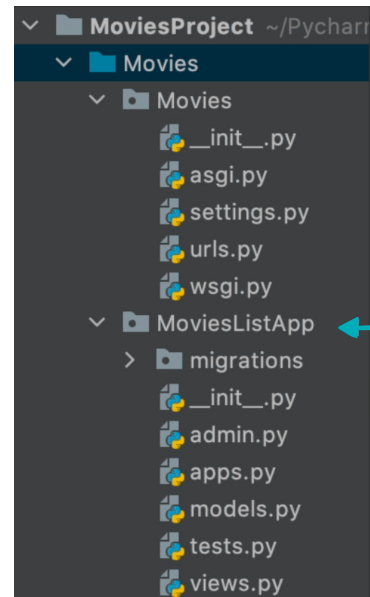


- ملف **settings.py** و يحتوي على الإعدادات الافتراضية الخاصة بالمشروع والتطبيقات الموجودة في Django.
- ملف **urls.py** وهو global URL mapping خاص لعمل URL mapping سواء لصفحات views أو child urls
- ملف **asgi.py** يحتوي الإعدادات لتشغيل المشروع على **ASGI (Asynchronous Server Gateway Interface)**.
-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

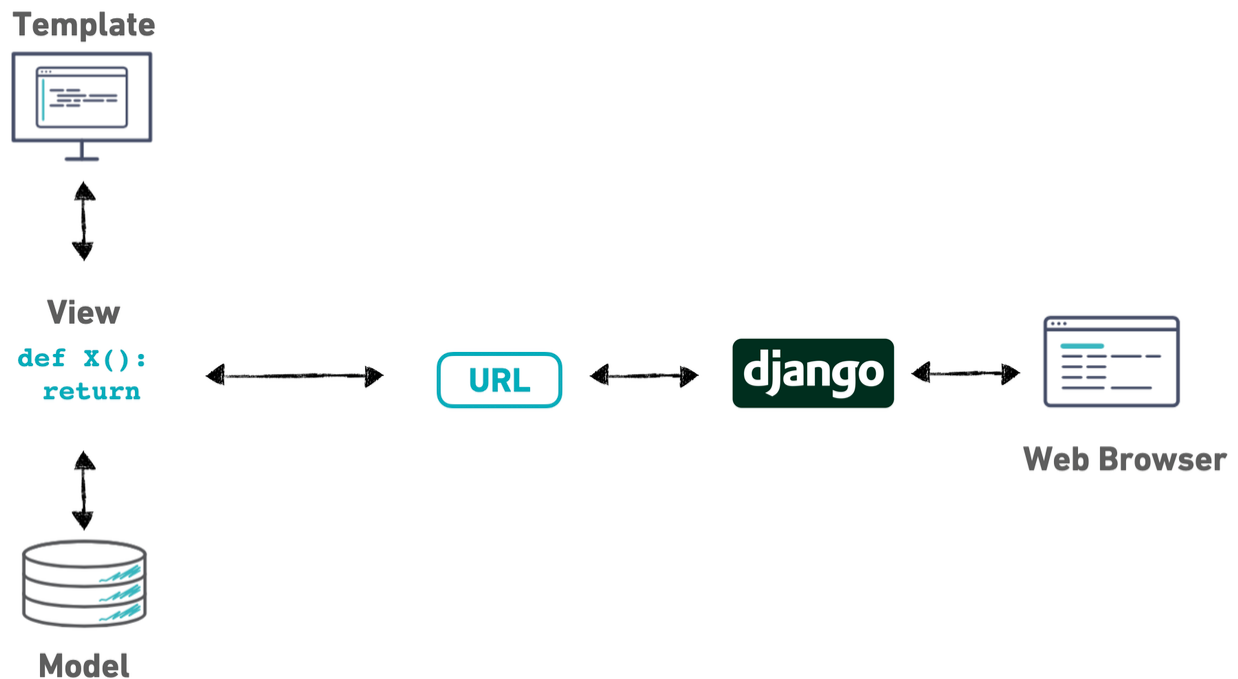
- ملف `admin.py` خاص بموقع `admin` والذي من خلاله يستطيع `admin` التعديل على البيانات عن طريق `Interface` جاهزة من `Django`.
- ملف `apps.py` يحتوى على `metadata` والإعدادات الخاصة بالتطبيق.
- ملف `models.py` نقوم بكتابة `model` بداخله لنتمكن من التعامل مع البيانات.
- ملف `views.py` نقوم بكتابة `Django 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 مثلاً: 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 ويتم إرساله إلى Web Server و يتكون HttpRequest من 4 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 ويستخدم لاسترجاع البيانات (Read) من remote page.
2. نوع POST ويستخدم لإضافة البيانات (Create) إلى remote page.
3. نوع PUT ويستخدم لتحديث البيانات (Update) في remote page.
4. نوع DELETE ويستخدم لحذف البيانات (Delete) من remote page.

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

ثانيًا: المسار أو Path الخاص بصفحة Web.

ثالثًا: 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 يقدّم بإرسال جزء خاص Body في المثال السابق وذلك لأننا فقط نقوم بزيارة للصفحة ولكن لو كنا نقوم بدخول صفحة Login مثلًا 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

أولاً: 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 (Found 302) و (301 Moved Permanently).
 4. 400-499 مجموعة الأرقام هذه تعني أنه لم تتم معالجة request بسبب مشكلة في (client-side) مثل (404 Not Found)، (401 Unauthorized)، (403 Forbidden).
 5. 500-599 مجموعة الأرقام هذه تعني أنه لم تتم معالجة request بسبب مشكلة في (server-side) مثل (500 Service Unavailable)، (502 Bad Gateway)، (504 Gateway Timeout)، (503 Internal Server Error).

ثانياً: Headers

يحتوي معلومات إضافية وهي كالتالي:


- Server: ويقوم 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 المرسل حيث تساعد Client باختيار طريقة عرض كل نوع من البيانات.
- Set-Cookie: تستخدم لتعيين cookie على المتصفح.

ثالثاً: 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:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

 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.

```
def BookInfo(request):  
    message = "...."  
    return HttpResponse(message)
```

النوع الثاني: class-based views وهنا نعبر عن view عن طريق كتابة python class ويمكن تطبيق جميع المفاهيم عليه مثل: class inheritance وغيره.

```
class BookInfo(TemplateView):  
    template_name = 'BookInfo.html'
```

أولاً: إنشاء Function-Based View

لإنشاء View قم باتباع الخطوات التالية:

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

```
from django.http import HttpResponse  
  
def Movie_Name(request):  
    return HttpResponse("This page displays information about the movies")
```

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

تطبيق URL Mapping

ملف urls.py يمثل global URL mapping خاص لعمل URL mapping سواء لصفحات views أو child .urls

عند فتح ملف settings.py سوف تلاحظ التالي:

```
ROOT_URLCONF = 'Movies.urls'
```

السطر السابق يمثل path وهو (Movies/urls.py) الذي نعود إليه عند تشغيل المشروع لعمل URL mapping.

الطريقة الأولى لتطبيق URL Mapping

لكتابة URL Mapping نتبع الطريقة التالية:

```
urlpatterns = [path('url-path/', views.my_view, name='my-view')]
```

```
urlpatterns = [path('BookList.html', views.BookList, name='BookList')]
```

- يمثل **BookList.html** الرابط الذي نقوم بكتابته بعد localhost بالشكل التالي:
(<http://127.0.0.1:8000/BookList.html>) وعند عدم كتابة أي اسم يكون الرابط (<http://127.0.0.1:8000>)
- يحدد **views.BookList** دالة view التي نقوم بالرجوع لها عند الدخول للرابط السابق.
- يعبر **name** عن اسم view.

في المثال السابق قمنا بكتابة View نقوم بطباعة جملة ولعرض هذه الصفحة للمستخدم لابد من تحديد عنوان URL الخاص بالصفحة، عن طريق الخطوات التالية:

- فتح ملف urls.py الموجود بداخل مجلد المشروع Movies وكتابة التالي.

```
import MoviesListApp.views  
path('', MoviesListApp.views.Movie_Name)
```

- نتأكد بأننا داخل مجلد Movies عن طريق كتابة cd Movies
- نقوم بتشغيل server عن طريق كتابة

```
python manage.py runserver
```

- يمكن تغيير عنوان الرابط عن طريق كتابة

```
path('Movies.html', MoviesListApp.views.Movie_Name)
```

ملاحظة : عند ظهور التحذير التالي " No Module named 'MoviesListApp' " عند السطر الذي يحوي
`import MoviesListApp.views`
قم بالضغط بالزر الأيمن على مجلد `Movies` الخارجي ثم اختر `make directory as sources root`

الطريقة الثانية لتطبيق URL Mapping

لنفرض أن لدينا مشروع eCommerce يحتوي عدة تطبيقات وكل تطبيق يحتوي عدة `views` على سبيل المثال:

eCommerce Project

StoreApp

- Welocme view
- AddtoCart view
- etc.

ReviewsApp

- Welocme view
- AddReview view
- etc.

عند عمل URL Mapping سوف يكون ملف `urls.py` بالشكل التالي:

```
import StoreApp.views
import ReviewsApp.views

path('Welocme.html', StoreApp.views.Welocme)
path('AddtoCart.html', StoreApp.views.AddtoCart)
path('Welocme.html', ReviewsApp.views.Welocme)
path('AddReview.html', ReviewsApp.views.AddReview)
```

ماذا لو كان لديك تطبيقات أكثر وكل منها تحتوي عدة `views`، في هذه الحالة نحتاج لتنظيم ملف `urls.py` عن طريق وضع ملف `urls.py` خاص بكل تطبيق عن طريق تنفيذ الأوامر التالية.

إضافة ملف `urls.py` خاص بتطبيق `MoviesListApp` عن طريق إنشاء ملف `urls.py` داخل المسار التالي:
`Movies/MoviesListApp` ونقوم بكتابة الأسطر التالية:

```
from django.urls import path
from . import views
urlpatterns = [path('', views.Movie_Name, name='Movie_Name')]
```

ثم بداخل ملف `urls.py` الموجود داخل مجلد `Movies` نقوم بتعديل `urlpatterns` كالآتي:

```
from django.urls import include
path('', include('MoviesListApp.urls')),
```

في المثال قمنا باستخدام `include function` حتى نفصل `urls` الخاصة بكل تطبيق عن التطبيقات الأخرى.

إنشاء Template

لنفرض أننا نريد كتابة `view` تقوم بطباعة عدد الأفلام، بداخل ملف `views.py` نقوم بكتابة التالي:

```
from django.http import HttpResponse
from .models import Movies_Info

def welcome_page(request):
    message = f"<html lang='en'>
<head>
<title>CSS Template</title>
<meta charset='utf-8'>
<meta name='viewport' content='width=device-width, initial-scale=1'>
<style>
* {
    box-sizing: border-box;
}

body {
    font-family: Arial, Helvetica, sans-serif;
}

/* Style the header */
header {
    background-color: #666;
```



```
padding: 30px;
text-align: center;
font-size: 35px;
color: white;
}

/* Create two columns/boxes that floats next to each other */
nav {
    float: left;
    width: 30%;
    height: 300px; /* only for demonstration, should be removed
*/
    background: #ccc;
    padding: 20px;
}

/* Style the list inside the menu */
nav ul {
    list-style-type: none;
    padding: 0;
}

article {
    float: left;
    padding: 20px;
    width: 70%;
    background-color: #f1f1f1;
    height: 300px; /* only for demonstration, should be removed
*/
}
```

```
/* Clear floats after the columns */
section::after {
    content: "";
    display: table;
    clear: both;
}

/* Style the footer */
footer {
    background-color: #777;
    padding: 10px;
    text-align: center;
    color: white;
}

/* Responsive layout - makes the two columns/boxes stack on to
p of each other instead of next to each other, on small screen
s */
@media (max-width: 600px) {
    nav, article {
        width: 100%;
        height: auto;
    }
}
</style>
</head>
<body>

<h2>CSS Layout Float</h2>
```

In this example, we have created a header, two columns/boxes and a footer. On smaller screens, the columns will stack on top of each other.

Resize the browser window to see the responsive effect (you will learn more about this in our next chapter - HTML Responsive.)

```
<header>
  <h2>Cities</h2>
</header>
```

```
<section>
  <nav>
    <ul>
      <li><a href="#">London</a></li>
      <li><a href="#">Paris</a></li>
      <li><a href="#">Tokyo</a></li>
    </ul>
  </nav>
```

```
<article>
  <h1>London</h1>
  <p>London is the capital city of England. It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p>
```

```
  <p>Standing on the River Thames, London has been a major settlement for two millennia, its history going back to its founding by the Romans, who named it Londinium.</p>
```

```
  </article>
</section>
```

```
<footer>
```

```

    <p>Footer</p>
</footer>

</body>
</html>"

    return HttpResponse(message)

```

من المثال السابق قمنا بكتابة أكواد html بداخل أكواد python وهذه الطريقة ليست عملية خصوصا في Django فمثلا لو كان لدينا صفحة html محتواها كبير، سوف يكون من الصعب قراءة الأكواد البرمجية لهذا السبب توفر Django طريقة تنظيمية عن طريق توفير templates التي تساعد بتنظيم ملفات html داخل مجلد خاص فيها.

يمثل Template أحد أهم أجزاء MVT ولإنشاءه لابد من إضافة مجلد Templates Directory و Base Template.

ثم اتباع الخطوات التالية:

- الضغط بالزر الأيمن على مجلد التطبيق MoviesListApp واختيار New Directory.
- إنشاء مجلد باسم "templates" ثم الضغط على OK.
- الضغط بالزر الأيمن على مجلد templates واختيار New file ثم إنشاء ملف base.html.
- بداخل ملف base.html نقوم بكتابة التالي:

```
<body> This is the template of MoviesListApp </body>
```

لو أردنا تغيير مكان مجلد Templates وقمنا بوضعه خارج مجلد التطبيق، عندها سوف نحتاج تعديل التالي: إضافة المجلد إلى ملف settings.py.

بداخل ملف settings.py نقوم بتعديل 'DIRS': [] وهي تمثل المجلدات التي تبحث Django فيها عن Templates. بشكل افتراضي 'DIRS': [] تكون فارغة و هذا يعني أن Django سوف يبحث عن مجلد TEMPLATES داخل مجلدات التطبيقات

```

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemp
lates',
        'DIRS': [],

```

نقوم بتعديل المسار كالتالي:

```
import os
```

```
'DIRS': [os.path.join(BASE_DIR, 'templates')],
```

ليش نستخدم رندر

ربط Templates مع View

بعد إنشائنا لمجلد templates وصفحات html، نقوم بربط صفحات html مع view الخاصة بها عن طريق الخطوات التالية:

- فتح ملف **views.py** الموجود في مجلد **MoviesListApp**
- نقوم بتعديل الملف كالتالي:

```
from django.http import HttpResponse  
from django.shortcuts import render  
def Movie_Name(request):  
    return render(request, "base.html")
```

في المثال السابق قمنا باستخدام **render function**: وهي function تقوم بإرجاع **HttpResponse** وتأخذ arguments التالية: الأول (**request** الذي يتم تمريره إلى **view**) والثاني (**template path**) وهنا يشير إلى **base.html** والثالث يمثل البيانات التي نريد إرسالها من **view** إلى **template**.

- تشغيل **server** عن طريق كتابة **python manage.py runserver**
- فتح صفحة **Movies.html** ونلاحظ ظهور المحتوى الموجود داخل ملف **base.html**

استخدام المتغيرات بداخل Templates

- يمكن تمرير متغيرات أو **Variables** من **Template** إلى **views** عن طريق تعديل ملف **base.html** كالتالي:

```
<body> This is the template of {{name}} </body>
```

- وأيضا تعديل ملف **views.py** كالتالي:

```
def Movie_Name(request):  
    name = "Harry Potter"  
    return render(request, "base.html", {"name": name})
```

لغة Django Template

أولاً: المتغيرات (Template Variables)

يمكن تمثيل المتغيرات بين أقواس بالشكل التالي: `{{ variable }}`
مثال:

```
template_variable = "Movies List App"

<body>
Welcome to {{ template_variable }}
</body>
```

عند عرض Template سوف يتم استبدال قيمة المتغير `template_variable` بكلمة `Movies List App` وبالتالي سوف يظهر لنا `Welcome to Movies List App`

ثانياً: الوسوم (Template Tags)

يعتبر tag مشابه لجمل `control flow` مثل `(if condition)` و `(for loop)`

يمكن تمثيل `for loop` بين أقواس وعلامة `%` بالشكل التالي: `{{ %for element in element_list% }}`
وأيضا نضيف نهاية جمل `control flow` بالشكل التالي: `{% endfor %}`
مثال:

```
<ul>
    {% for item in Movies_list %}
        <li>
            <span> Movie Name: </span> <span>{{ item.Movie.name }}</span>
        </li>
    {% endfor %}
</ul>
```

عند عرض Template سوف تظهر لنا قائمة بجميع أسماء الأفلام الموجودة
أيضا يمكن تمثيل `if condition` بين أقواس وعلامة النسبة المئوية بالشكل التالي: `{% if item in list %}` ثم نضيف نهاية `if condition` التالي: `{% endif %}` مثال:

```
{% if item.Movie.name == "Frozen" %}
<span> This movie produced by Disney.</span>
```

```
{% endif %}
```

عند عرض Template سوف تظهر لنا عبارة This movie produced by Disney عند طباعة فلم "Frozen".

ثالثاً: التعليقات (Comments)

يمكن تمثيل التعليقات بين أقواس وعلامة النسبة المئوية بالشكل التالي : قبل بداية التعليق نكتب `{% comment %}` وعند نهايته نكتب `{% endcomment %}`
مثال:

```
{% comment %}  
    <p>This text has been commented out</p>  
{% endcomment %}
```

عند عرض Template لن تظهر لنا جملة This text has been commented out

رابعاً: FILTERS

تستخدم في تغيير طريقة عرض المتغيرات (variable) وصياغتها بطرق أخرى ويمكن تمثيل FILTERS بالشكل التالي: `{{ variable|filter }}` حيث يتم فصل المتغير (variable) عن اسم filter بالعلامة (|)
مثال:

```
{{ variable|lower }}: This converts the variable string into lowercase.  
{{ variable|title }}: This converts the first letter of every word into uppercase.
```