تكليف بحث عن :

DJANGO TEMPLATES ENGINE
THE REMAINING FILTERS FOR
DJANGO
ADD NEVW FILTER

المهندس: مالك المصنف

م: مالك هدوان

محرك قوالب Django (Django Templates Engine)

تعريف محرك القوالب

محرك قوالب Django هو نظام يُستخدم لإنشاء صفحات ويب ديناميكية عن طريق دمج البيانات مع قوالب HTML. يعتمد على مبدأ فصل المنطق البرمجي عن التصميم، مما يسهل على المطورين والمصممين العمل بشكل منفصل.

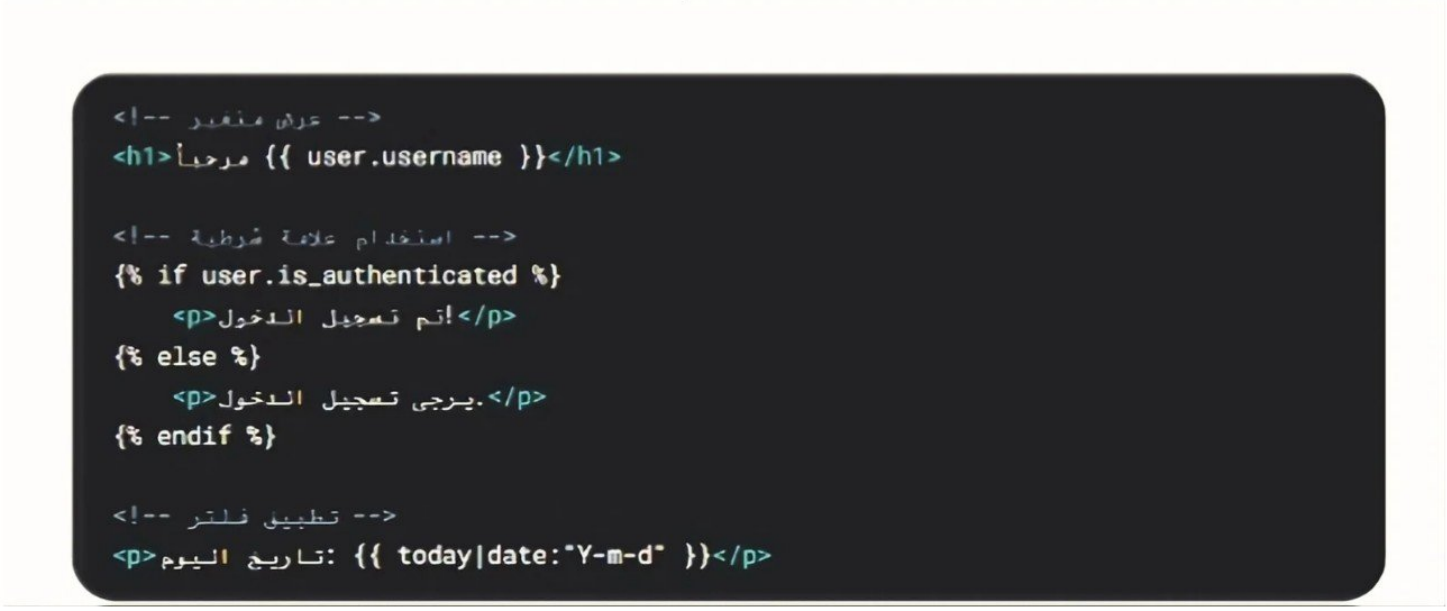
مكونات محرك القوالب

يتكون محرك القوالب في Django من:

* المتغيرات (Variables): تُعرض باستخدام {{ variable }}.
* العلامات (Tags): تنفذ عمليات برمجية مثل {% for %} أو {% if %}.
* الفلاتر (Filters): لتعديل قيم المتغيرات مثل {{ name|upper }}.
* التوريث (Template Inheritance): نستخدم {% extends %} لإعادة استخدام القوالب.

مميزات محرك قوالب Django

* الأمان: يحمي من هجمات XSS (Cross-site Scripting) عبر الهروب التلقائي من الأحرف الخطرة.
* المرونة: يدعم تكوين قوالب مخصصة وإنشاء فلاتر مخصصة.
* الأداء العالي: يستخدم تخزين مؤقت للقوالب (Template Caching) لتحسين السرعة.



ما هي الفلاتر في **Django**؟

الفلاتر هي دوال تُستخدم لمعالجة المتغيرات داخل القوالب. تأتي **Django** مع مجموعة كبيرة من الفلاتر الجاهزة، مثل:

الفلتر	الوصف	مثال	
lower	يحول النص إلى أحرف صغيرة	`{{ "HELLO"	lower }}` → "hello"
upper	يحول النص إلى أحرف كبيرة	`{{ "hello"	upper }}` → "HELLO"
length	يعيد طول القائمة أو النص	`{{ list	length }}`	
slice	يقطع جزءا من النص أو القائمة	`{{ "Django"	slice:":3" }}` → "Dja"
date	ينسق التاريخ	`{{ today	date:"Y-m-d" }}`	
default	يعيد قيمة افتراضية إذا كانت القيمة فارغة	`{{ value	default:"N/A" }}`	

فلاتر متقدمة في **Django**

* **safe:** يعطل الهروب التلقائي للأحرف الخطرة (يُستخدم بحذر).
* **truncatechars:** يقطع النص بعد عدد محدد من الأحرف.
* **join:** يجمع عناصر قائمة بنص فاصل.
* **json_script:** يحول القاموس إلى **JSON** آمن للاستخدام في **JavaScript**.

```
<!-- قطع النص بعد 50 حرفاً -->
<p>{{ article.content|truncatechars:50 }}</p>

<!-- ربط قائمة بفاصلة -->
<p>الألوان: {{ colors|join:", " }}</p>

<!-- عرض JSON -->
{{ data|json_script:"data-json" }}
<script>
    const data = JSON.parse(document.getElementById('data-json').textContent);
</script>
```

إضافة فلاتر جديدة (ADD NEW FILTER)

أحيانًا تحتاج إلى معالجة بيانات بطريقة غير موجودة في الفلاتر الافتراضية.
لحسن الحظ، يمكن بسهولة إنشاء فلاتر مخصصة في **Django.**

خطوات إنشاء فلتر مخصص:

الخطوة **1:** إنشاء مجلد **templatetags** داخل التطبيق:

```
myapp/
├── __init__.py
├── models.py
├── templatetags/
│   ├── __init__.py
│   └── custom_filters.py  # ملف الفلاتر المخصصة
└── views.py
```

الخطوة **2:** كتابة الفلتر المخصص في **custom_filters.py**

```
from django import template

register = template.Library()

@register.filter(name='add_prefix')
def add_prefix(value, prefix):
    return f"{prefix}-{value}"

@register.filter
def reverse_string(value):
    return value[::-1]
```

الخطوة **3:** تحميل الفلتر في القالب واستخدامه:

```
load custom_filters %}
Hello-Django --> :الناتج
<p>{{
"Django"|add_prefix:"Hel
<lo" }}</p
nohtyP --> <p>{{ :الناتج
"Python"|reverse_string
<}}</p
```

3.3 أمثلة متقدمة للفلاتر المخصصة

* فلتر لحساب النسبة المئوية:

```
<!-- end list -->
register.filter@
:def percentage(value, total)
"%return f"{(value / total) * 100:.2f}

:html
{{ percentage:200|50 }} <-- النسبة<p>
*** <p/>
```

* فلتر لحساب النسبة المئوية:

* فلتر لترتيب القوائم:

```
<!-- end list -->
@register.filter
def sort_list(value, attribute=None):
if attribute:
return sorted(value, key=lambda x:
getattr(x, attribute))
return sorted(value)
```