SoftwareEngineering

Assignment (3)

Created By :
Dheya Al-Hadhrami

Supervisor :
Eng. Malik Al-Mossanif

Introduction

(Django)

في قلب إطار عمل Django يكمن النمط المعماري Model-View-Template (MVT)،

وهو متغير من النمط الأكثر شهرة Model-View-Controller (MVC).

في هذا النموذج، تلعب طبقة "القالب" (Template) دورًا حاسمًا:

فهي مسؤولة عن عرض البيانات. محرك القوالب (Template Engine)هو الآلية التي

تشغل هذه الطبقة، مما يتيح فصل بيانات التطبيق ومنطق الأعمال

(الذي يتم التعامل معه بواسطة النماذج (Models) والعروض (Views))

عن عرضه النهائي.

Assignment Objectives

- تحليل مقارن لمحركات قوالب Python لـ Django .

- مراجعة منهجية لفلاتر لغة قوالب Django (DTL) .

- دليـــــل عملي لتنفيذ فـــــلاتر DTL مخصصة .

Part I:

A Comparative Analysis of Python Template Engines for Django

Chameleon Mako Jinja2 **DTL**

Django Template Language (DTL)

فلسفته الأساسية هي فرض فصل صارم بين منطق العرض ومنطق الأعمال. إنه ليس Python مضمنًا في HTML، بل لغة خاصة بالقوالب تهدف إلى البساطة وتقييد المنطق المعقد داخل القوالب.

بنية بسيطة

يستخدم `{{ variable }}` للمتغيرات و `{% tag %}` للتحكم.

```
{{ post.title }}

Published by {{ post.author.name }}
```

وراثة القوالب

ميزة قوية تسمح بإنشاء قوالب أساسية وإعادة استخدامها.

```
{% extends "base.html" %}
{% block content %}...{% endblock %}
```

فلاتر مدمجة

مجموعة غنية من الفلاتر لتنسيق البيانات.

```
{{ post.publish_date|date:"F d, Y" }}
```

Chameleon Mako **Jinja2** DTL

Jinja2

يسعى لتحقيق توازن بين قوة DTL المحدودة وحرية المحركات الأخرى. يوفر ميزات أكثر قوة وبيئة معزولة (sandboxed) آمنة، مما يجعله خيارًا شائعًا وقويًا.

قوة معززة

يسمح باستدعاء الدوال مع وسائط مباشرة من القالب.

```
{{ user.get_profile(section='details') }}
```

وحدات الماكرو

لإنشاء مكونات HTML قابلة لإعادة الاستخدام.

```
{% macro input(name, value='', type='text') %}...{% endmacro %}
```

أداء عالٍ

يتم تجميعه إلى bytecode مما يجعله أسرع بكثير من DTL.

```
{# Jinja2 templates compile for speed #}
```

Chameleon **Mako** Jinja2 DTL

Mako

يعتمد على مبدأ "Python هي لغة برمجة نصية رائعة". يوفر القوة الكاملة لـ Python داخل القوالب، مما يلغي القيود ولكنه يزيد من المسؤولية الأمنية على المطور.

Python كامل

يمكنك كتابة أي كود Python داخل كتل `<% ... %>`.

```
<%
  x = 10
  y = x * 2
%>
Value is: ${y}
```

كتل قابلة للاستدعاء

تعريف دوال Python مباشرة في القالب باستخدام `<%def>`.

```
<%def name="my_func(x)">
  return "Hello %s" % x
```

أداء ممتاز

مثل Jinja2، يتم تجميعه إلى bytecode للحصول على أقصى سرعة.

```
## Mako is known for its high performance
```

Chameleon Mako Jinja2 DTL

Chameleon

فلسفته هي أن يكون "غير تدخلي". المنطق مدمج في سمات HTML، مما يحافظ على القالب كمستند HTML صالح يمكن فتحه في أدوات التصميم الرسومية.

لغة السمات (TAL)

يتم التحكم في المنطق عبر سمات `tal:`.

```
Page Title
```

التكرار

تكرار العناصر باستخدام `tal:repeat`.

```
- Item
```

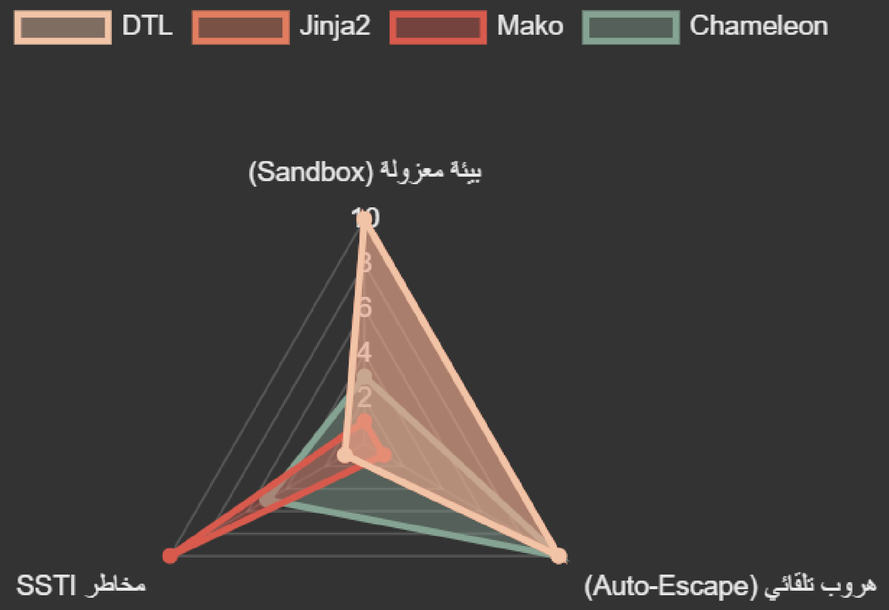
صديق للمصممين

تبقى القوالب ملفات HTML صالحة يمكن معاينتها في المتصفح.

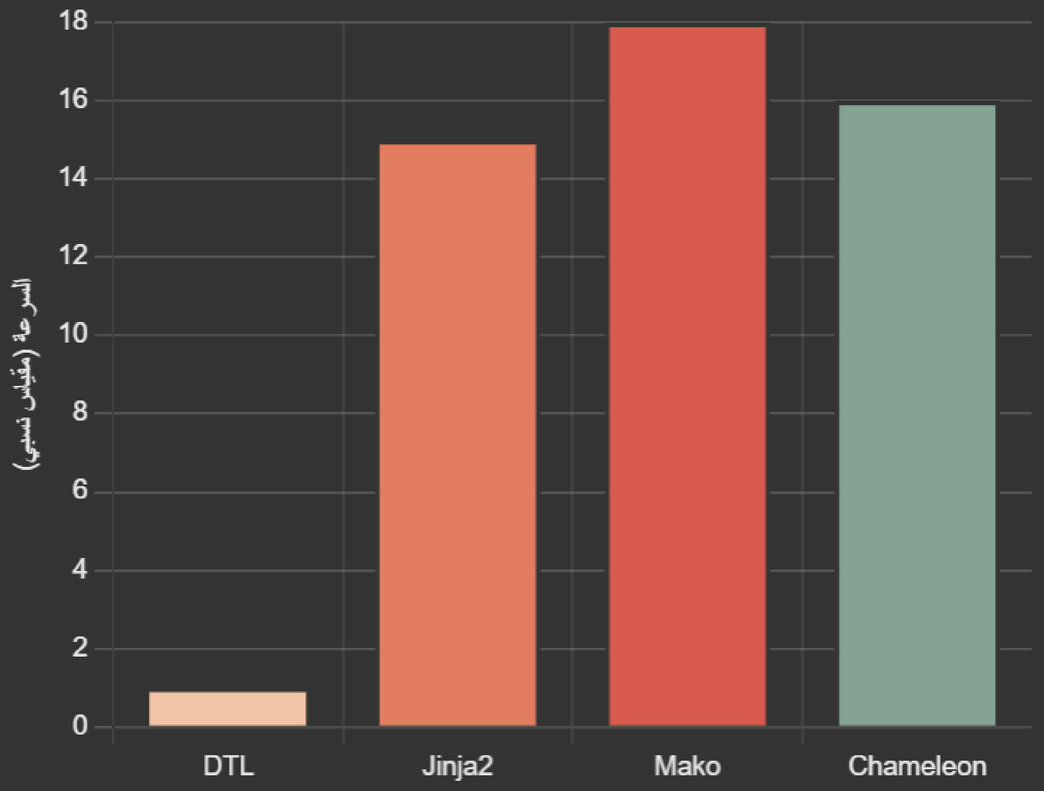
```
Admin controls
```

Chameleon Mako Jinja2 DTL

تقييم الوضع الأمني



مقارنة الأداء (سرعة التصيير)



Part II :

A Systematic Review of Django Template Language (DTL) Filters

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

dictsort	default	date	capfirst	add
lower	length	json_script	join	filesizeformat
timesince	striptags	slugify	safe	pluralize
	wordcount	urlize	upper	truncatechars

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

filesizeformat	add

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

capfirst | lower | slugify | striptags | truncatechars

upper | wordcount

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

date | timesince

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

default | pluralize

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

dictsort | join | length

الكل | رقمي | نصي | تاريخ/وقت | شرطي | قائمة | أمان/HTML

json_script | safe | urlize

Part III :

A Practical Guide to Implementing Custom DTL Filters

الخطوة 1: بنية المشروع

يجب وضع الفلاتر المخصصة في بنية مجلدات محددة داخل تطبيق Django.

الخطوة 2: كتابة منطق الفلتر

كتابة دالة Python التي تقوم بالتحويل المطلوب وتسجيلها كفلتر.

الخطوة 3: الاستخدام في القالب

تحميل مكتبة الفلاتر وتطبيق الفلتر الجديد على المتغيرات.

الخطوة 1: بنية المشروع

لإنشاء فلاتر مخصصة، يجب أن يتبع تطبيق Django الخاص بك بنية محددة. أنشئ مجلدًا باسم `templatetags` داخل تطبيقك، ويجب أن يحتوي هذا المجلد على ملف `__init__.py` فارغ وملف Python لمرشحاتك (على سبيل المثال، `custom_filters.py`).

```
myapp/
├── __init__.py
├── models.py
├── views.py
└── templatetags/
    ├── __init__.py
    └── custom_filters.py
```

الخطوة 1: بنية المشروع

يجب وضع الفلاتر المخصصة في بنية مجلدات محددة داخل تطبيق Django.

الخطوة 2: كتابة منطق الفلتر

كتابة دالة Python التي تقوم بالتحويل المطلوب وتسجيلها كفلتر.

الخطوة 3: الاستخدام في القالب

تحميل مكتبة الفلاتر وتطبيق الفلتر الجديد على المتغيرات.

الخطوة 2: كتابة منطق الفلتر

في ملف `custom_filters.py`، قم باستيراد مكتبة القوالب، وأنشئ مثيلاً لها، ثم استخدم المزخرف `@register.filter` لتسجيل دالتك كفلتر.

```
from django import template
import math

register = template.Library()

@register.filter(name='reading_time')
def reading_time(value, wpm=200):
    word_count = len(str(value).split())
    minutes = math.ceil(word_count / wpm)
    if minutes < 1:
        return "أقل من دقيقة"
    return f"{minutes} دقائق قراءة"
```

الخطوة 1: بنية المشروع

يجب وضع الفلاتر المخصصة في بنية مجلدات محددة داخل تطبيق Django.

الخطوة 2: كتابة منطق الفلتر

كتابة دالة Python التي تقوم بالتحويل المطلوب وتسجيلها كفلتر.

الخطوة 3: الاستخدام في القالب

تحميل مكتبة الفلاتر وتطبيق الفلتر الجديد على المتغيرات.

الخطوة 3: الاستخدام في القالب

أخيرًا، في القالب الخاص بك، قم أولاً بتحميل مكتبة الفلاتر المخصصة باستخدام وسم `{% load %}`، ثم طبق الفلتر باستخدام رمز الأنبوب `|`.

```
{% load custom_filters %}

<h1>{{ post.title }}</h1>

<p>
  وقت القراءة: {{ post.content|reading_time }}
</p>
```

Thank You

Looking forward to seeing you
in the second assignment ;)