Django Template Language

Templates dynamisch gestalten

Was sind Templates?

templates sind meist HTML-Dateien, die mit der Django-Template-Sprache mit dynamischen Inhalten gefüllt werden können.

Es können aber auch json-Dateien, csv-Dateien und vieles mehr als Template genutzt werden.

Ein Template beinhaltet also statischen HTML-Code und dynamische Bereiche, die in diesen Code "hinein gerendert" werden.

Variablen und Tags

Ein Template beinhaltet Variablen, die ersetzt werden, wenn das Template gerendert wird.

Die Namen dieser Variablen entsprechen den Werten der Objektinstanz, die über den Key des Kontexts erreichbar ist.

Ein Template beinhaltet auch sogenannte Tags, die die Logik innerhalb eines Templates kontrollieren. Dazu gehören u.a. Loops und Bedingungen.

Variablen ersetzen

Wenn die Template-Engine im Template eine Variable findet, versucht sie, diese zu ersetzen.

Gelingt ihr das nicht, wird standardmäßig ein leerer String ersetzt.

Dh. es kommt zu keinem Fehler, wenn der zu ersetzende Werte, dh. der Variablenname nicht existiert.

Variablen

Variablen haben folgendes Schema:

{{ variablen_name }}

Beispiele:

{{ company_instanz.name }}

Variablen Beispiel view

Das Beispiel auf der vorherigen Seite könnte folgende View haben:

```
def single(request, id):
    company = Company.objects.get(pk=id)

    return render(request, 'company/single.html', {
        'company_instanz': company,
        'num': 42,
    })
```

Das Dict mit den Objektinstanz-Values wird Kontext genannt. Wir könnten hier alles übergeben, was wir wollen. ZB. auch einen beliebigen numerischen Wert.

Variablen Beispiel Template

Aus Platzgründen nur der Body-Bereich des Templates (ohne head).

```
<body>
<h1>Firma: {{ company_instanz.name }}</h1>
<p>{{ company_instanz.sub_title }}</p>
<p>numerischer Wert: {{ num }}
</body>
```

Variablen: Filter

Häufig müssen Daten aus der Anwendungsschicht in etwas anderer Form präsentiert werden. Zum Beispiel muss ein Floatwert, welcher in der Datenbank mit 5 Nachkommastellen gehalten wird, auf der Website (Präsentationsebene) mit nur einer Nachkommastelle präsentiert werden.

Diese Aufgabe gehört zur Präsentationsebene und sollte daher im Template und nicht in der view erledigt werden.

Hierzu bietet Django die Möglichkeit, Variablen zu filtern.

Wie funktionieren Filter?

Ein einfaches Beispiel als Demonstration:

```
<h1>{{ company|upper }}</h1>
```

Rechts von der Variable wird anhand des Pipe-Symbols der Filter eingeleitet. Danach folgt der Name des Filters (Django beinhaltet ca. 50 verschiedene Filter).

Filter sind verkettbar. Wir können mit der Pipe nach jeder Filteroperation wieder einen neuen Filter auf das Ergebnis der vorherigen Operation anwenden.

```
<h1>{{ company|upper|truncatechars:20 }}</h1>
```

Beispiel Filter im Template

Aus Platzgründen nur der Body-Bereich des Templates (ohne head).

```
<body>
<h1>Firma: {{ company_instanz.name|upper }}</h1>
<p>{{ company_instanz.sub_title }}</p>
<p>numerischer Wert: {{ num }}
</body>
```

Wichtige Filter

Name	Aufgabe	Beispiel
upper	Großbuchstaben	{{ company.name|upper }}
lower	Kleinbuchstaben	{{ company.name|lower }}
default	Default-Wert wenn nicht vorhanden	{{ company.sub_title|default:'nicht vorhanden' }}
length	String- oder Listenlänge	{{ company.name|length }}
linebreaks	Wandelt newlines in HTML-Absätze um	{{ company.description|linebreaks }}
slice	Wie der Slice-Operator in Python	{{ company.name|slice: ":20" }}
truncatewords	Nach Anzahl an Wörtern beenden	{{ company.description|truncatewords:10 }}
truncatechars	Wie truncatewords, nur mit Zeichen	{{ company.description|truncatechars:10 }}

Übersicht aller Filter

https://docs.djangoproject.com/en/3.1/ref/templates/builtins/#ref-templates-builtins-filters

TAGS

Tags steuern die Logik des Templates und gehören damit zur Präsentationslogik.

Das Schema von Tags ist:

{% tag_name %}

Tags sind komplexer als Variablen. Einige kontrollieren den Programmfluss, andere dienen vorranging der Template-Strukturierung. Manche Tags benötigen deshalb einen Start- und einen End-Tag.

{% tag %} ... tag contents ... {% endtag %}

Tag: der for-Loop

Der For-Loop ähnelt dem Python For-Loop. Über jedes Interierbare (zb. Sequenzen, Iteratoren) kann wie in Python mit einem for-loop iteriert werden.

In jeder Iteration steht dann der Wert als Variable zur Verfügung:

```
{% for company in companies %}
    {{ company }}
{% endfor %}
```

```
{%  for char in 'abcde' %}
   {{ char }}
{% endfor %}
```

Tag: if, else

Mit if-else lassen sich auch im Template bedingte Anweisungen schreiben:

```
{% if companies %}
    Anzahl an Firmen: {{companies|length}}
{% else %}
    keine Firmen vorhanden!
{% endif %}
```

Tag: if, elif, else

```
{% if companies|length  >= 5 %}
    viele Firmen
{% elif companies|length > 3 %}
    mittelviele Firmen
{% else %}
    wenige Firmen
{% endif %}
```

Tag: Template Hierachie

Templates werden anhand sogenannter Blöcke hierachisch organisiert.

Die Grunddatei base.html enhält dazu alle HTML-Inhalte, die auf jeder Seite des Projektes vorkommen.

Die Unterseiten erweitern (=extend) das base-Template und laden ihrerseits Content-Blöcke in die dafür im Base-Template angelegten Platzhalter. Dafür wird der Tag {% extends %} genutzt.

Dafür gibt es die Block-Tag, welches ein öffnendes und schließendes Tag hat:

{% block content %} {% endblock %}

Beispiel: Base.html = definiert block content

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Beispiel einer Base.html</title>
</head>

<body>
<div>
{% block content %}
{% endblock %}
</div>
</body>
```

Beispiel: home.html erweitert base.html

```
{% extends 'company/base.html' %}

{% block content %}
<p>
Dieser Text wird jetzt in den Content-Block von base gerendert!
</p>

{% endblock %}
```

Tag: URL

Niemals sollten innerhalb von Templates URLs hardkodiert in den Quelltext geschrieben werden!

Um eine URL als Hyperlink in einem HTML-Dokument (=Template) zu notieren, wird die umgekehrte URL-Auflösung genutzt.

Jede View hat einen eindeutigen Eintrag in den Urlpatterns der App. Dort wird auch ein name dieser URL vergeben:

```
urlpatterns = [
    path('companies', views.overview, name='overview'),
]
```

Tag: Url

Diesen Namen der Route können wir nun nutzen, um im Template darauf zu referenzieren. Durch Doppelpunkt vom App-Namen getrennt steht der Namen der in den Urlpatterns defienierten Route.

```
<a href="{% url 'company:overview'  %}">Alle Companies</a>
```