09 - Django Templates: Vererbung, Includes und Statische Dateien

Einleitung

- **Themen**: Effiziente Erstellung konsistenter Webseiten-Layouts durch Template-Vererbung (`{% extends %}`, `{% block %}`). Wiederverwendung von Template-Fragmenten mit `{% include %}`. Grundlagen der Einbindung von statischen Dateien (CSS, JavaScript, Bilder) mit dem `{% static %}`-Tag.
- **Fokus**: Reduzierung von Code-Duplizierung in Templates, Erstellung einer einheitlichen Seitenstruktur und die Grundlagen zur Einbindung von Styling und clientseitiger Logik.
- **Lernziele**:
  - Das Konzept der Template-Vererbung verstehen und anwenden können.
  - Basis-Templates erstellen und Blöcke definieren, die von Kind-Templates überschrieben werden.
  - Die Nützlichkeit von `{{ block.super }}` erkennen.
  - Wiederverwendbare Template-Teile mit `{% include %}` einbinden.
  - Statische Dateien (insbesondere CSS) korrekt einrichten und in Templates laden.

1. Template-Vererbung: Das DRY-Prinzip in Templates

Das DRY-Prinzip ("Don't Repeat Yourself") ist auch in der Template-Erstellung wichtig. Template-Vererbung erlaubt es, eine Basis-HTML-Struktur zu definieren, die von allen (oder vielen) Seiten der Webanwendung wiederverwendet wird.

- **Das Basis-Template (`base.html`)**: Dies ist die Vorlage, die das Grundgerüst der Seite enthält (z.B. `<html>`, `<head>`, `<body>`, Navigation, Footer). In diesem Basis-Template werden **Blöcke** (`{% block ... %}`) definiert. Diese Blöcke dienen als Platzhalter, die von "Kind-Templates" (Child Templates) mit spezifischem Inhalt gefüllt werden können.

  Beispiel für `myapp/templates/myapp/base.html`:

```
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Meine Webseite{% endblock %}</title>
    </head>
<body>
    <header>
        <h1>Meine Webseite</h1>
        <nav>
            </nav>
    </header>

    <main>
        {% block content %}
        <p>Willkommen auf meiner Seite!</p>
        {% endblock %}
    </main>

    <footer>
        <p>{% block footer_content %}© 2025 Meine Webseite{% endblock %}</p>
    </footer>
</body>
</html>
```

- **Das Kind-Template (Child Template)**: Ein Kind-Template **erbt** vom Basis-Template mit dem `{% extends "myapp/base.html" %}`-Tag (dies muss die allererste Zeile im Template sein). Es kann dann die im Basis-Template definierten Blöcke überschreiben.

  Beispiel für `myapp/templates/myapp/home.html`:

```
{% extends "myapp/base.html" %}

{% block title %}Startseite – Meine Webseite{% endblock %}
```

```
{% block content %}
    <h2>Willkommen auf der Startseite!</h2>
    <p>Dies ist der spezifische Inhalt für die Startseite.</p>
{% endblock %}

{# Der footer_content Block wird nicht überschrieben und verwendet den Standard aus
base.html #}
```

- `{{ block.super }}`: Manchmal möchte man den Inhalt eines Blocks aus dem Eltern-Template nicht komplett ersetzen, sondern ergänzen. Mit `{{ block.super }}` kann der Inhalt des Blocks aus dem Eltern-Template innerhalb des überschreibenden Blocks im Kind-Template eingefügt werden.

  Beispiel im Kind-Template `about.html`:

  ```
  {% extends "myapp/base.html" %}

  {% block title %}Über Uns{% endblock %}

  {% block footer_content %}
      {{ block.super }} <p>Kontaktieren Sie uns unter info@example.com</p>
  {% endblock %}
  ```

2. Wiederverwendung mit `{% include %}`

Der `{% include "template_name.html" %}`-Tag erlaubt es, den Inhalt eines anderen Templates an der Stelle des Include-Tags einzufügen. Dies ist nützlich für wiederverwendbare Teile wie Navigationsleisten, Sidebars oder Footer, die in mehreren Templates identisch sind, aber nicht Teil der Hauptvererbungsstruktur sein sollen.

Beispiel `myapp/templates/myapp/partials/navigation.html`:

```
<nav>
    <ul>
        <li><a href="{% url 'myapp:home' %}">Start</a></li>
        <li><a href="{% url 'myapp:about' %}">Über Uns</a></li>
        </ul>
</nav>
```

Einbindung in `base.html`:

```
    ...
    <body>
        <header>
            <h1>Meine Webseite</h1>
            {% include "myapp/partials/navigation.html" %}
        </header>
    ...
```

Man kann dem inkludierten Template auch Variablen mit dem `with`-Argument übergeben:

```
{% include "myapp/partials/user_greeting.html" with user_name="Max" %}
```

3. Statische Dateien (CSS, JavaScript, Bilder)

Statische Dateien sind jene Dateien, die nicht vom Server dynamisch generiert werden, sondern direkt ausgeliefert werden. Dazu gehören CSS-Dateien für das Styling, JavaScript-Dateien für clientseitige Interaktivität und Bilder.

- **Konfiguration in `settings.py`**:
    - `STATIC_URL`: Die URL, unter der die statischen Dateien im Browser erreichbar sein werden (z.B. `/static/`).

      ```
      # settings.py
      STATIC_URL = '/static/'
      ```

    - `STATICFILES_DIRS` (optional, aber oft nützlich): Eine Liste von zusätzlichen Verzeichnissen, in denen Django nach statischen Dateien suchen soll (z.B. ein projektweiter `static`-Ordner).

      ```
      import os

      STATICFILES_DIRS = [
          os.path.join(BASE_DIR, 'static'), # Projektweiter static-Ordner
      ]
      ```

    - `STATIC_ROOT` (für Deployment): Der Pfad zu einem einzelnen Verzeichnis, in das alle statischen Dateien mit dem Befehl `python manage.py collectstatic` kopiert werden, um sie im Produktivbetrieb von einem Webserver ausliefern zu lassen. Dies ist für die Entwicklung meist nicht relevant.
- **Speicherort**: Ähnlich wie bei Templates sucht Django mit `APP_DIRS=True` (Standard für Templates) auch in einem `static`-Unterordner jeder App nach statischen Dateien. Es ist gute Praxis, hier wieder einen App-spezifischen Unterordner anzulegen (z.B. `myapp/static/myapp/css/style.css`).
- **Verwendung in Templates mit `{% static %}`**: Zuerst muss der `static`-Tag am Anfang des Templates geladen werden: `{% load static %}`.

  Beispiel in `base.html`:

  ```
  {% load static %}
  <!DOCTYPE html>
  <html lang="de">
  <head>
      <meta charset="UTF-8">
      <title>{% block title %}Meine Webseite{% endblock %}</title>
      <link rel="stylesheet" href="{% static 'myapp/css/style.css' %}">
  </head>
  <body>
      ...
      <img src="{% static 'myapp/images/logo.png' %}" alt="Logo">
      ...
      <script src="{% static 'myapp/js/script.js' %}"></script>
  </body>
  </html>
  ```

  Der `{% static %}`-Tag generiert den korrekten URL-Pfad zur statischen Datei, basierend auf der `STATIC_URL`-Einstellung.

Fazit

- **Template-Vererbung (`extends`, `block`)**: Schafft konsistente Layouts und reduziert Code-Wiederholung, indem eine Basisstruktur definiert wird, deren Teile von Kind-Templates angepasst werden können. `{{ block.super }}` erlaubt das Ergänzen geerbter Inhalte.
- **Includes (`include`)**: Dienen der Wiederverwendung kleinerer, in sich geschlossener Template-Fragmente.
- **Statische Dateien (`static`)**: Unverzichtbar für Design und Interaktivität. Der `{% static %}`-Tag sorgt für korrekte Pfade zu CSS, JavaScript und Bildern.
- **Struktur**: Eine gute Organisation von Template- und Static-Ordnern (oft mit App-spezifischen Unterordnern) ist wichtig für die Übersichtlichkeit.

Projekt-Anwendung (Leitfaden-Projekt)

Für das "Online-Umfragesystem" (`Polls`-Projekt) wird eine `base.html` erstellt und die bestehenden Templates (`question_list.html`, `question_detail.html`) werden angepasst, um davon zu erben.

1. `polls/templates/polls/base_polls.html` **erstellen:**

```
{% load static %}
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Polls App{% endblock %}</title>
    <link rel="stylesheet" href="{% static 'polls/style.css' %}">
</head>
<body>
    <header>
        <h1><a href="{% url 'polls:list' %}">Polls Anwendung</a></h1>
        </header>
    <main>
        {% block content %}{% endblock %}
    </main>
    <footer>
        <p>&copy; Polls Inc.</p>
    </footer>
</body>
</html>
```

2. `polls/static/polls/style.css` **erstellen (Beispiel):**

```
/* polls/static/polls/style.css */
body { font-family: sans-serif; margin: 20px; background-color: #f4f4f4; }
header h1 a { color: #333; text-decoration: none; }
main { background-color: #fff; padding: 15px; border-radius: 5px; }
ul { list-style-type: none; padding: 0; }
li a { text-decoration: none; color: #007bff; }
footer { margin-top: 20px; text-align: center; font-size: 0.9em; color: #777; }
```

3. `question_list.html` **anpassen:**

```
{% extends "polls/base_polls.html" %}

{% block title %}Alle Umfragen - Polls App{% endblock %}

{% block content %}
    {% if latest_question_list %}
        <ul>
        {% for question in latest_question_list %}
            <li>
                <a href="{% url 'polls:detail' question.id %}">{{ question.question_text }}</a>
                (Veröffentlicht: {{ question.pub_date|date:"d. F Y" }})
            </li>
        {% endfor %}
        </ul>
    {% else %}
        <p>Keine Umfragen verfügbar.</p>
    {% endif %}
{% endblock %}
```

4. `question_detail.html` **anpassen:**

```
{% extends "polls/base_polls.html" %}

{% block title %}{{ question.question_text }} - Polls App{% endblock %}

{% block content %}
    <h2>{{ question.question_text }}</h2>
    <p>Veröffentlicht am: {{ question.pub_date|date:"D, d. M. Y, H:i" }} Uhr</p>

    {% if question.choice_set.all %}
        <ul>
        {% for choice in question.choice_set.all %}
            <li>{{ choice.choice_text }} -- {{ choice.votes }} Stimme(n)</li>
        {% endfor %}
        </ul>
    {% else %}
        <p>Für diese Frage gibt es noch keine Antwortmöglichkeiten.</p>
    {% endif %}

    <p><a href="{% url 'polls:list' %}">Zurück zur Übersicht</a></p>
{% endblock %}
```

Damit die statische CSS-Datei gefunden wird, muss sichergestellt sein, dass die App `polls` in `INSTALLED_APPS` ist und `APP_DIRS: True` in der `TEMPLATES`-Einstellung gesetzt ist (für Templates) sowie die `STATIC_URL` korrekt konfiguriert ist (für Static Files).

Cheat Sheet

- **Template-Vererbung**:
  - `{% extends "basis_template.html" %}` (erste Zeile im Kind-Template)
  - `{% block blockname %}` ... `{% endblock %}` (im Basis- und Kind-Template)
  - `{{ block.super }}` (um Inhalt des Eltern-Blocks einzufügen)
- **Include**:
  - `{% include "partial_template.html" %}`
  - `{% include "partial.html" with var=wert %}`
- **Statische Dateien**:
  - `{% load static %}` (am Anfang des Templates)
  - `{% static 'pfad/zur/datei.css' %}`
  - `settings.py`: `STATIC_URL = '/static/'`
  - `collectstatic`-Befehl für Deployment: `python manage.py collectstatic`

Übungsaufgaben

1. **Eigene Basis-HTML erstellen**:
   - Ein neues, einfaches Django-Projekt mit einer App `main` erstellen.
   - In `main/templates/main/` eine `base.html`-Datei erstellen mit mindestens einem `title`-Block und einem `content`-Block. Es sollte auch einen einfachen Header und Footer enthalten.
2. **Kind-Templates erstellen**:
   - Zwei Kind-Templates erstellen: `home.html` und `contact.html`, die beide von `base.html` erben.
   - Jedes Kind-Template soll die Blöcke `title` und `content` mit spezifischem Inhalt füllen.
3. **Navigation als Include**:
   - Eine separate `navigation.html`-Datei erstellen, die eine einfache Linkliste enthält (z.B. zur Home- und Kontaktseite – die URLs können zunächst mit `#` als Platzhalter dienen).
   - Diese `navigation.html` in den Header der `base.html` includieren.
4. **Einfache CSS-Datei einbinden**:
   - Im `main`-App-Verzeichnis einen Ordner `static/main/css/` erstellen und darin eine `style.css`.
   - In der `style.css` einige einfache Regeln definieren (z.B. Hintergrundfarbe für den `body`, andere Schriftart für `h1`).
   - Diese `style.css` korrekt in die `base.html` einbinden.
   - Views und URLs erstellen, um die `home.html` und `contact.html` im Browser aufrufen zu können und das Ergebnis zu prüfen.

Schüler-Projekt (Eigenständig): Community Recipe Sharing Platform

Die "Community Recipe Sharing Platform" erhält nun eine konsistente Grundstruktur und ein erstes Styling.

Aufgabe:

1. **Basis-Template `base_recipes.html` erstellen**:
   - Im Ordner `recipes/templates/recipes/` eine Datei `base_recipes.html` anlegen.
   - Diese soll das Grundgerüst für alle Seiten der Rezeptplattform bilden:
     - HTML-Grundstruktur (`<!DOCTYPE html>`, `<html>`, `<head>`, `<body>`).
     - Einen `{% block title %}` im `<head>`.
     - Einen einfachen Header mit dem Namen der Plattform (z.B. "Meine Rezeptwelt") und einer Platzhalter-Navigation.
     - Einen Hauptinhaltsbereich mit `{% block content %}`.
     - Einen einfachen Footer mit einem Copyright-Hinweis.
2. **Partials für Navigation erstellen (Optional, aber empfohlen)**:
   - Im Ordner `recipes/templates/recipes/partials/` eine Datei `_navigation.html` erstellen.
   - Darin eine `<ul>`-Liste mit Links zu "Alle Rezepte" (später die Listenansicht) und "Rezept hinzufügen" (später das Formular) definieren. (Die URLs können zunächst Platzhalter wie `#` sein oder auf die bereits erstellten benannten URLs zeigen, falls vorhanden).
   - Diese `_navigation.html` in den Header der `base_recipes.html` includieren.
3. **Bestehende Templates (`recipe_list.html`, `recipe_detail.html`) anpassen**:
   - Die bereits in Skript 2.3 erstellten Templates `recipe_list.html` und (optional) `recipe_detail.html` so anpassen, dass sie von `base_recipes.html` erben.
   - Den `title`-Block und den `content`-Block in jedem dieser Kind-Templates mit dem spezifischen Inhalt füllen.
4. **Statische CSS-Datei einrichten und einbinden**:
   - Im `recipes`-App-Verzeichnis die Ordnerstruktur `static/recipes/css/` erstellen.
   - Darin eine Datei `main_style.css` anlegen.
   - In `main_style.css` einige grundlegende CSS-Regeln definieren (z.B. für `body`, Überschriften, Links, oder ein einfaches Layout für die Rezeptliste).
     - *Tipp für schnelles Styling*: Ein sehr einfaches CSS-Framework wie Pico.css oder Simple.css kann über ein CDN eingebunden werden, um schnell ein ansprechendes Grunddesign zu erhalten, ohne viel eigenes CSS schreiben zu müssen. Dies kann alternativ oder zusätzlich zum eigenen CSS erfolgen.
   - Die `main_style.css` (oder das CDN-CSS) korrekt im `<head>` der `base_recipes.html` einbinden, indem der `{% static %}`-Tag verwendet wird (nicht vergessen: `{% load static %}` am Anfang des Templates).
5. **Testen**: Den Entwicklungsserver starten und die Rezeptliste (und ggf. Detailseite) aufrufen. Überprüfen, ob die Vererbung funktioniert, die Navigation sichtbar ist und das CSS angewendet wird.