03 - Django: Views und URL-Parameter

Einleitung

- Dieses Kapitel vertieft das Verständnis von Django Views als Herzstück der Anfragenverarbeitung.
- Es werden verschiedene Möglichkeiten der Rückgabe von Webantworten beleuchtet.
- Ein Schwerpunkt liegt auf dem Auslesen dynamischer Daten aus URLs mittels Pfad-Konvertern.
- Dies ermöglicht die Erstellung von interaktiven Webanwendungen, die auf spezifische Eingaben in der URL reagieren.

---

1. Django Views im Detail

Eine View-Funktion ist eine Python-Funktion, die eine eingehende Webanfrage entgegennimmt (repräsentiert durch ein `HttpRequest`-Objekt) und eine Webantwort zurückgibt (ein `HttpResponse`-Objekt oder eine seiner Unterklassen). Views sind der Ort, an dem die Geschäftslogik einer Webanwendung hauptsächlich stattfindet.

1.1 Das `HttpRequest`-Objekt

Wenn ein Browser eine Anfrage an den Django-Server sendet, verpackt Django die Details dieser Anfrage in ein `HttpRequest`-Objekt. Dieses Objekt wird dann als erstes Argument an die View-Funktion übergeben.

Wichtige Attribute und Methoden des `HttpRequest`-Objekts:

- **`request.method`**: Ein String, der die HTTP-Methode der Anfrage angibt (z.B. `'GET'`, `'POST'`).
- **`request.GET`**: Ein `QueryDict`-Objekt, das alle HTTP GET-Parameter (aus dem Query-String der URL) enthält.
- **`request.POST`**: Ein `QueryDict`-Objekt, das alle HTTP POST-Parameter (aus dem Request Body) enthält.
- **`request.path`**: Der vollständige Pfad zur angeforderten Seite, exklusive des Domainnamens und Query-Strings.
- **`request.user`**: Ein `User`-Objekt, das den aktuell angemeldeten Benutzer repräsentiert (wird später im Zusammenhang mit Authentifizierung behandelt).
- **`request.META`**: Ein Python-Dictionary mit allen verfügbaren HTTP-Headern.

1.2 Rückgabe von Webantworten (`HttpResponse` und `render`)

View-Funktionen müssen immer ein `HttpResponse`-Objekt zurückgeben. Django bietet hierfür verschiedene Möglichkeiten:

1.2.1 `HttpResponse`

Die einfachste Form ist die direkte Rückgabe von Text oder HTML-Code.

Verwendung:

```
# myapp/views.py
from django.http import HttpResponse

def simple_text_view(request):
    return HttpResponse("Dies ist ein einfacher Text als Antwort.")

def simple_html_view(request):
    html_content = "<h1>Willkommen!</h1><p>Dies ist HTML von der View.</p>"
    return HttpResponse(html_content)
```

Diese Methode ist nützlich für sehr einfache Antworten oder zum Debugging, aber selten für komplette Webseiten, da die direkte HTML-Erstellung im Python-Code unpraktisch ist.

1.2.2 `render()`

Die Funktion `render()` ist der Standardweg, um HTML-Templates in Django zu verwenden. Sie kombiniert ein gegebenes Template mit einem Kontext-Dictionary und gibt ein `HttpResponse`-Objekt zurück, das den gerenderten Text enthält.

Vorteile von `render()`:

- **Trennung von Logik und Darstellung:** Der Python-Code (View) bleibt von der HTML-Struktur (Template) getrennt, was die Wartung und Entwicklung erleichtert.
- **Dynamische Inhalte:** Daten können aus der View an das Template übergeben und dort dynamisch angezeigt werden.

Verwendung: Um `render()` nutzen zu können, muss zuerst ein Template-Verzeichnis eingerichtet werden, was wir in einem späteren Skript detailliert behandeln werden. Für den Moment nehmen wir an, dass Django weiß, wo es Templates finden kann.

```
# myapp/views.py
from django.shortcuts import render # 'render' importieren

def greet_with_name_view(request):
    # Ein Dictionary, das Daten für das Template bereitstellt
    context = {
        'name': 'Welt'
    }
    # Rendert das Template 'my_template.html' mit dem bereitgestellten Kontext
    return render(request, 'my_app/my_template.html', context)
```

Hierbei würde `my_template.html` eine Variable `{{ name }}` verwenden, um den Wert aus dem Kontext-Dictionary anzuzeigen.

2. URL-Routing mit Pfad-Konvertern

In Django kann nicht nur ein fester URL-Pfad einer View zugeordnet werden, sondern auch dynamische Teile in der URL, sogenannte **URL-Parameter**, ausgelesen werden. Dies wird durch **Pfad-Konverter** in den URL-Patterns ermöglicht.

2.1 Was sind URL-Parameter?

URL-Parameter sind dynamische Segmente innerhalb eines URL-Pfads, die als Variablen an die View-Funktion übergeben werden. Sie ermöglichen es, spezifische Ressourcen oder Aktionen über die URL zu identifizieren.

Beispiel: Statt `http://example.com/products/view/1` und `http://example.com/products/view/2` mit zwei separaten Routen, kann eine einzige Route `http://example.com/products/view/<product_id>/` verwendet werden, wobei `<product_id>` ein Parameter ist.

2.2 Pfad-Konverter in `django.urls.path`

Die `path()`-Funktion von Django verwendet Pfad-Konverter, um die dynamischen Teile einer URL zu erfassen und in den richtigen Python-Datentyp zu konvertieren, bevor sie an die View-Funktion übergeben werden.

Syntax: `<konverter_name:parameter_name>`

Standard-Pfad-Konverter:

- **`str`**: (Standard, wenn kein Konverter angegeben ist) Passt zu jeder Nicht-leeren Zeichenkette, exklusive `/`.
  - Beispiel: `<str:username>`
- **`int`**: Passt zu einer ganzen Zahl.
  - Beispiel: `<int:year>`
- **`slug`**: Passt zu einer "Slug"-Zeichenkette (ASCII-Zeichen, Zahlen, Bindestriche oder Unterstriche). Wird oft für suchmaschinenfreundliche URLs verwendet.
  - Beispiel: `<slug:post_slug>`
- **`uuid`**: Passt zu einer UUID (Universally Unique Identifier).
  - Beispiel: `<uuid:book_id>`
- **`path`**: Passt zu jeder Nicht-leeren Zeichenkette, inklusive `/`. Kann verwendet werden, um den Rest eines URL-Pfads zu erfassen.
  - Beispiel: `<path:file_path>`

2.3 URL-Parameter in Views auslesen

Die in den URL-Patterns definierten Parameter werden als zusätzliche Argumente an die View-Funktion übergeben, und zwar in der Reihenfolge, in der sie im URL-Pattern erscheinen.

Schritt-für-Schritt-Anleitung:

1. **View-Funktion erstellen:** Definieren Sie eine View-Funktion in Ihrer App (`myapp/views.py`), die Argumente entgegennimmt.

```
# myapp/views.py
from django.http import HttpResponse

def greet_user(request, name):
    response_text = f"Hallo, {name}! Willkommen auf unserer Seite!"
    return HttpResponse(response_text)

def multiply(request, a, b):
    result = a * b
    response_text = f"Das Ergebnis von {a} * {b} ist {result}."
    return HttpResponse(response_text)
```

2. **App-spezifische URLs anpassen (`myapp/urls.py`):** Fügen Sie die URL-Patterns mit Pfad-Konvertern hinzu.

```
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('greet/<str:name>/', views.greet_user, name='greet_user'),  # String-Parameter
    path('calculate/multiply/<int:a>/<int:b>/', views.multiply, name='multiply_numbers'), #
Integer-Parameter
]
```

3. **Haupt-URL-Konfiguration überprüfen (`myproject/urls.py`):** Stellen Sie sicher, dass die App-URLs weiterhin korrekt in die Haupt-URLs eingebunden sind.

```
# myproject/urls.py
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('my_app_prefix/', include('myapp.urls')), # Beispiel-Präfix
]
```

   (Hinweis: `my_app_prefix/` sollte durch Ihr tatsächliches Präfix ersetzt werden, z.B. `app/` aus Skript 02.)

4. **Server starten und testen:**
   - Starten Sie den Entwicklungsserver: `python manage.py runserver`
   - Testen Sie im Browser die Funktionalität:
     - Begrüßung: `http://127.0.0.1:8000/my_app_prefix/greet/Alex/`
     - Multiplikation: `http://127.0.0.1:8000/my_app_prefix/calculate/multiply/5/3/`

Wichtigkeit von URL-Parametern: URL-Parameter sind essentiell für die Erstellung von "sauberen" und semantischen URLs. Sie ermöglichen es, eindeutige Adressen für individuelle Ressourcen (z.B. `/products/123/` für Produkt mit ID 123) oder spezifische Aktionen zu schaffen, die direkt im Pfad der URL ausgedrückt werden.

Fazit

- Django Views verarbeiten Anfragen und geben Antworten zurück; das `HttpRequest`-Objekt bietet detaillierte Informationen über die Anfrage.
- Antworten können direkt über `HttpResponse` oder effizienter über `render()` mit Templates erzeugt werden.
- URL-Parameter in Kombination mit Pfad-Konvertern ermöglichen die flexible und dynamische Übergabe von Daten über den URL-Pfad an View-Funktionen.

- Diese Konzepte sind grundlegend für den Aufbau interaktiver und ressourcenorientierter Webanwendungen in Django.

Cheat Sheet: Skript 03 - Django Views und URL-Parameter

`HttpRequest` Objekt (Auswahl)

- `request.method`: `'GET'`, `'POST'`, etc.
- `request.GET`: GET-Parameter aus Query-String.
- `request.POST`: POST-Parameter aus Request Body.
- `request.path`: Pfad der angefragten URL.
- `request.user`: Aktuell angemeldeter Benutzer.

Rückgabe von Antworten

- **Direkte Text/HTML-Antwort:**

```
from django.http import HttpResponse
return HttpResponse("Mein Text")
```

- **Rendern eines Templates (empfohlen):**

```
from django.shortcuts import render
context = {'key': 'value'}
return render(request, 'app_name/template.html', context)
```

URL-Parameter mit Pfad-Konvertern

- **Syntax:** `<konverter_name:parameter_name>`
- **View-Definition:** Argumente in View-Funktion empfangen: `def my_view(request, param1, param2):`
- **Standard-Konverter:**
  - `str`: Zeichenkette (Standard).
  - `int`: Ganze Zahl.
  - `slug`: "Slug"-Zeichenkette (alphanumerisch, Bindestriche, Unterstriche).
  - `uuid`: UUID-String.
  - `path`: Beliebige Zeichenkette inklusive Schrägstriche.

Beispiel `urls.py` mit Parametern

```
# myapp/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('user/<str:username>/', views.show_user_profile, name='user_profile'),
    path('article/<int:article_id>/', views.show_article, name='article_detail'),
]
```

Übungsaufgaben zu Skript 03

Ziel der Übungen: Vertiefung des Verständnisses von Django Views und der dynamischen Übergabe von Daten über URL-Parameter.

1. **Dynamische Begrüßung mit URL-Parameter:**
   - **Aufgabe:**

1. Erstellen Sie in Ihrer `homepage` App (oder einer neuen Test-App) eine View-Funktion namens `begruessung_nach_name`.
2. Diese View soll einen String-Parameter `name` aus der URL entgegennehmen.
3. Die View soll eine `HttpResponse` zurückgeben, die den übergebenen Namen verwendet, z.B. "Hallo, [Name des Benutzers]! Schön, Sie zu sehen.".
4. Definieren Sie ein entsprechendes URL-Pattern in `homepage/urls.py` (oder der App-`urls.py`), das den Pfad `'greet/<str:name>/'` der View zuordnet. Geben Sie der Route den Namen `'begruessung'`.
5. Testen Sie die Funktion, indem Sie URLs wie `http://127.0.0.1:8000/web/greet/Anna/` und `http://127.0.0.1:8000/web/greet/Peter/` aufrufen.

2. **Einfache Rechenoperation mit Integer-Parametern:**

   - **Aufgabe:**
     1. Erstellen Sie eine View-Funktion namens `summe` in derselben App.
     2. Diese View soll zwei Integer-Parameter `num1` und `num2` aus der URL entgegennehmen.
     3. Die View soll die Summe der beiden Zahlen berechnen und als `HttpResponse` zurückgeben, z.B. "Die Summe von 10 und 5 ist 15.".
     4. Definieren Sie ein URL-Pattern in der App-`urls.py` für den Pfad `'sum/<int:num1>/<int:num2>/'`. Geben Sie der Route den Namen `'summe'`.
     5. Testen Sie mit verschiedenen Zahlenkombinationen (z.B. `http://127.0.0.1:8000/web/sum/10/5/`).

3. **Fehleranalyse: Falscher Pfad-Konverter:**

   - **Aufgabe:** Ändern Sie im URL-Pattern für die `summe`-View den Typ eines Parameters von `int` zu `str` (z.B. `path('sum/<int:num1>/<str:num2>/', ...)`).
   - Rufen Sie die URL `http://127.0.0.1:8000/web/sum/10/5/` erneut auf.
   - Welchen Fehler erhalten Sie nun? Erklären Sie, warum dieser Fehler auftritt und warum der korrekte Pfad-Konverter wichtig ist.
   - Korrigieren Sie den Fehler wieder.

🚀 Schüler-Projekt: Aufbau der "Community Recipe Sharing Platform"

Konzept für den heutigen Tag: Die "Community Recipe Sharing Platform" wird heute durch die Anwendung von URL-Parametern flexibler. Die bisherige statische Willkommensseite wird um dynamische Elemente erweitert. Obwohl wir noch keine Datenbankmodelle für Rezepte haben, können wir simulieren, wie die Plattform auf Anfragen für spezifische Rezepte reagieren würde.

Aufgabe (Tag 3):

1. **"Rezept-Detail"-View implementieren (simuliert):**

   - Öffnen Sie die `views.py` Ihrer `recipes`-App im Projekt `recipe_platform`.
   - Fügen Sie eine neue View-Funktion namens `recipe_detail` hinzu.
   - Diese Funktion soll einen Integer-Parameter namens `recipe_id` entgegennehmen.
   - Die View soll eine `HttpResponse` zurückgeben, die eine Nachricht wie "Details für Rezept mit ID: [recipe_id] werden hier angezeigt." enthält.
   - *Noch keine Datenbankinteraktion!* Es geht nur darum, den Parameter aus der URL auszulesen und zu verwenden.

2. **URL-Routing für Rezept-Details anlegen:**

   - Öffnen Sie die `urls.py` Ihrer `recipes`-App (`recipe_platform/recipes/urls.py`).
   - Fügen Sie ein neues URL-Pattern hinzu, das den Pfad `'<int:recipe_id>/'` der `recipe_detail`-View-Funktion zuordnet. Geben Sie dieser Route den Namen `'detail'`.
   - Stellen Sie sicher, dass dieses Pattern nach der "Willkommen"-Route (falls sie auch einen leeren Pfad `''` nutzt) definiert ist, um Überschneidungen zu vermeiden, oder nutzen Sie ein eindeutiges Präfix wie `'recipes/<int:recipe_id>/'`. Im Zweifel ist es besser, die "Home"-Route auf einen expliziten Pfad wie `'welcome/'` zu setzen und die Detail-Route auf `'<int:recipe_id>/'`.

3. **Funktionalität überprüfen:**

   - Stellen Sie sicher, dass der Django-Entwicklungsserver läuft.
   - Testen Sie die neue Funktionalität, indem Sie URLs wie `http://127.0.0.1:8000/recipes/1/`, `http://127.0.0.1:8000/recipes/42/` oder `http://127.0.0.1:8000/recipes/999/` im Browser aufrufen.

- Überprüfen Sie, ob die korrekte ID in der Antwort angezeigt wird.

Ziele für das Schüler-Projekt heute: Das Konzept der URL-Parameter und Pfad-Konverter praktisch anwenden, um dynamische Seiten in der Recipe Sharing Platform zu simulieren und die Grundlagen für zukünftige Rezept-Detailansichten zu legen.