Django - Prezentacja danych

Zajęcia Prymus 2020

Agnieszka Rudnicka, rudnicka@agh.edu.pl

Szablony HTML w Django

Każda strona WWW ma kilka stałych elementów interfejsu, takich jak: menu, stopka, obszar gdzie wyświetlane są treści, czasem pasek boczny i inne. Każda podstrona naszej aplikacji musi więc składać się z tego samego kodu, kopiowanego wszędzie i trudnego w utrzymaniu. Gdy zrobimy gdzieś literówkę, to musimy poprawiać we wszystkich plikach z osobna! Gdy chcemy dodać coś nowego do interfejsu też.

Na szczęście istnieją frameworki! Django jak wiele innych narzędzi ma sposób na taką redundancję kodu - Django HTML Templating Engine. W skrócie nazywane dalej szablonami HTML Django.

Do tej pory mieliśmy do czynienia głównie ze standardową składnią HTML.

Django wzbogaca możliwości HTML między innymi o dziedziczenie szablonów. To pozwala nam napisać jeden raz szablon bazowy naszej aplikacji, który będzie dynamicznie załączany do każdej podstrony, która po nim dziedziczy. Brzmi magicznie? Nie szkodzi, zobaczmy jak to wygląda w praktyce.

Szablon bazowy dla naszej aplikacji

Na dobry początek utwórzmy w katalogu `books/templates/` nowy plik `base.html`. Będzie to nasz bazowy szablon wykorzystywany przez inne podstrony. Jeśli na poprzednich zajęciach przygotowaliście swoje strony z wykorzystaniem Boostrapa, to jest dobre miejsce, żeby przenieść ten kod.

Zacznijmy od czegoś prostego (proponowana początkowa zawartość `base.html`):

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <title>Moja biblioteka</title>
</head>
<body>

</body>
</html>
```

Powyższy HTML wyświetli nam jedynie pustą stronę z tytułem karty "Moja biblioteka". Dodajmy więc przynajmniej jakiś nagłówek (oczywiście w środku elementu `<body> </body>`):

```
<h1>
    Witamy w bibliotece!
</h1>
```

W ten sposób będziemy wiedzieli, że każda strona wykorzystuje ten kod HTML, bo nam się wyświetli nagłówek.

Następnie dodajmy HTMLowy tag Django:

```
{% block content %}
    Tu będzie treść...
{% endblock %}
```

Ten fragment kodu informuje Django, że w tym miejscu pliku HTML można wstawić jakąś zawartość w szablonach HTML, które rozszerzają ten. Kontynuujmy...

Tag `{% block %}{% endblock %}` może być pusty, ale jeśli coś w nim umieścimy zadziała to jak domyślna wartość.

W szablonie z poprzednich zajęć `hello.html` dodajmy na górze pliku, w pierwszej linijce `extends` .

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

<h1>Witaj świecie z HTML!</h1>
```

A następnie wejdźmy na tę stronę w przeglądarce. Rezultaty? :)

To teraz zobaczymy co oznaczało "wstawić jakąś zawartość w szablonach HTML, które rozszerzają ten (base.html)". Zmieńmy kod na następujący.

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

{% block content %}
    <h1>Witaj świecie z HTML!</h1>
{% endblock %}
```

I ponownie udajmy się do przeglądarki.

Prezentacja danych z bazy

W tej chwili mamy działający model na książki oraz narzędzie, które pozwala na zarządzanie nimi (panel administracyjny). Jednak chcielibyśmy pochwalić się zawartością naszej biblioteczki światu. W tym celu oczywiście nie rozdamy wszystkim haseł dostępu, lub go nie zdejmiemy, to byłoby niebezpieczne!

Spróbujmy zatem wyświetlić listę książek korzystając z szablonów HTML Django. To one pozwalają nam dynamicznie dodawać treść pobraną z bazy danych (a w międzyczasie zmienianą przez użytkowników w panelu administracyjnym).

Widok listy książek

Udajmy się do `books/views.py` aby utworzyć widok listy książek.

Na dobry początek trzeba zaimportować model (mądre IDE jak Pycharm same to zaproponują i zrobią). Następnie piszemy kolejną funkcję, która przyjmuje zapytanie (`request`) jako argument.

```
from books.models import Book

def list_books(request):
    books = Book.objects.all()
    return render(request, template_name="book_list.html", context={"books": books})
```

W tym miejscu wiele się dzieje! Zacznijmy od góry:

- dzięki zaimportowaniu modelu `Book` będziemy mogli wejść w interakcje z bazą danych za pośrednictwem Django, czyli między innymi:
  - dodawać nowe obiekty typu `Book`
  - edytować i usuwać istniejące obiekty `Book`
  - odczytywać a także filtrować istniejące obiekty `Book`
- `objects` - każdy model w Django ma coś co się nazywa **manager**, nie będziemy wchodzić tutaj w szczegóły, ale jest to interfejs przez który dostarczane są nam operacje na bazie danych. W praktyce, umożliwia to nam tworzenie, edycję, usuwanie i inne zapytania do bazy danych.
- `Book.objects.all()` użyje menedżera obiektów typu `Book` i zapyta bazę danych o WSZYSTKIE książki. Dostaniemy więc listę całej zawartortości naszej biblioteki.
- Do wygenerowania odpowiedzi aplikacji zostanie użyty szablon `book_list.html` (tak, musimy go teraz stworzyć!)
- Lista książek zostanie dodana do kontekstu HTML

Rejestracja podstrony listy książek

Aby widok mógł być wyświetlony trzeba mu przydzielić jakiś adres. Udajmy się więc do `urls.py` czyli o URL resolvera naszego projektu. Tutaj należy dodać `path()`, który będzie kierował zapytania przeglądarek użytkowników z konkretnego adresu na stronę powstającej listy książek.

Przykładowo, do listy `urlpatterns` dodajmy linijkę:

```
urlpatterns = [
    ...

    # http://127.0.0.1:8000/ksiazki
    path("ksiazki", views.list_books),  # NEW
]
```

Teraz link http://127.0.0.1:8000/ksiazki powinien nas kierować do listy książek, którą przed chwilą napisaliśmy w `views.py`. Został jeszcze jeden element - szablon HTML, który wyświetli dane.

Szablon HTML listy

Stwórzmy plik o nazwie `book_list.html` (oczywiście musi być umieszczony w katalogu szablonów naszej aplikacji `books/templates/`).

Na początek możemy sprawdzić, czy wszystko dobrze zaprogramowaliśmy. Spróbujmy więc wypisać zmienną `books`, którą dodaliśmy do kontekstu szablonu.

```
{{ books }}
```

Jeśli w przeglądarce pojawiła się lista obiektów z tajemniczym `QuerySet`, to jesteśmy na dobrej drodze.

Aby wypisać elementy list prostą pętlą można posłużyć się tagiem Django `{% for %}`.

```
{% for book in books %}
<p>
    Książka: {{ book }}
</p>
{% endfor %}
```

Powyższy kawałek kodu prze-iteruje po liście, którą dostarczyliśmy z widoku. Następnie dla każdego elementu wstawi HTML zawarty w środku, czyli w naszym przypadku akapit z napisem `Książka {{ book }}`.

Dalsze prace

- Wypisz osobno każde z pól modelu książki np. tytuł, datę publikacji
- Wykorzystaj HTML i CSS do poprawienia wyglądu listy
- Przebuduj szablon `book_list.html` tak aby wykorzystywał (rozszerzał) szablon bazowy `base.html`
- Dodaj widok szczegółów pojedynczej książki