Kurs rozszerzony języka Python

Środowisko Django, cz. 3

Marcin Młotkowski

19 stycznia 2018

Plan wykładu

Plan wykładu

Klasa SimpleTestCase

SimpleTestCase

Alternatywna (djangowa) klasa do testów z dodatkowymi asercjami.

Dodatkowe asercje

assertHTMLEqual(html1, html2)

Porównanie dwóch plików html; ignoruje białe znaki, kolejność atrybutów.

assertInHTML(needle, haystack)

Wyszukiwanie fragmentu w html'u.

assertTemplateUsed(response, template)

Sprawdzenie, czy użyto do wygenerowania odpowiedzi wskazanego szablonu

I inne (np. formularze).

Plan wykładu

Przykładowa aplikacja

System Zapisy, modele

- Wykladowca
- Student
- Wyklad

Przypomnienie

Modele

Typy danych.

Co można robić z wartościami

- zapamiętywanie/odtwarzanie
- edycja/tworzenie

Edycja wartości

- ręczne zbudowanie formularza w html'u;
- zdefiniowanie obiektu (pod)klasy `forms.Form`;
- skorzystanie z klasy dedykowanej.

Konstrukcja formularzy dla modeli

```
from django.forms import ModelForms
from  wyklad.zapisy.models import Wyklad

class WykladForm(ModelForm):
    class Meta:
        model = Wyklad
        fields = ['tytul', 'opis', 'ects']
```

Co mamy w wyniku

model

```
class Wyklad(models.Model):
        nazwa = models.CharField(max_length=140)
        ects = models.IntegerField()
        wykladowca = models.ForeignKey(Wykladowca)
```

Ręczny formularz

```
class WykladForm(forms.Form):
    nazwa = forms.CharField(max_length=140)
    ects = forms.IntegerFields()
    wykladowca =
        forms.ModelChoiceField(queryset=Wykladowca.objects.all())
```

Definiowanie pól do edycji: uzupełnienie

Edycja wszystkich pól

```
class WykladForm:
    class Meta:
        model = Wyklad
        fields = ['__all__']
```

Zmiana domyślnych ustawień

```
class WykladForm(ModelForm):
    class Meta:
        model = Wyklad
        fields = ['tytul', 'opis']
        widgets = {
                'opis' : Textarea(attrs={'cols': 80, 'rows': 20})
        }
```

Przypomnienie

ModelForm.is_valid()

Zapis danych

Obiekty klasy `ArticleForms` implementują metodę `save()`

Scenariusze

Nowy obiekt

```
w = WykladForm(request.POST)
nowy_wyklad = w.save()
```

Scenariusze

Nowy obiekt

```
w = WykladForm(request.POST)
nowy_wyklad = w.save()
```

Istniejący obiekt

```
w = Wyklad.objects.get(id=13)
w = WykladForm(request.POST, instance=w)
nowy_wyklad = w.save()
```

.save() waliduje formularz.

Plan wykładu

Autentykacja

django oferuje domyślnie

- system użytkowników;
- system uprawnień;
- formularze i widoki do logowania.

Użytkownicy

Utworzenie użytkowników

Z panelu administracyjnego

Użytkownicy

Utworzenie użytkowników

Z panelu administracyjnego

podstawowe atrybuty modelu User

- username
- password
- email
- first_name, last_name

Hasła

Hasła są przechowywane w postaci hasza, z losową solą i informacją o algorytmie haszowania.

Zmiana hasła

```
user = User.objects.get(...)
user.set_password('noweHaslo')
user.save()
```

Weryfikacja użytkownika

Autentykacja

```
from django.contrib.auth import authenticate
user = authenticate(username='mm', password='123456')
if user is not None:
    ...
```

Weryfikacja użytkownika

Autentykacja

```
from django.contrib.auth import authenticate
user = authenticate(username='mm', password='123456')
if user is not None:
    ...
```

Weryfikacja

```
if request.user.is_authenticated():
    # zrób coś
else :
    # anonimowy użytkownik
```

logowanie

Autentykacja tylko weryfikuje, ale nie tworzy sesji dla użytkownika.

logowanie

Autentykacja tylko weryfikuje, ale nie tworzy sesji dla użytkownika.

```
from django.contrib.auth import authenticate, login

def login_view(request):
    username = request.POST['username']
    password = request.POST['password']
    user = authenticate(username=username, password=password)
    if user is not None:
        if user.is_active:
            login(request, user)
            # Przekieruj na odpowiednią stronę.
        else :
            # informacja o zablokowanym koncie
        else :
            # Zgłoś błąd logowania.
```

Wylogowanie

```
from django.contrib.auth import logout
def logout_view(request):
    logout(request)
```

Rozwiązanie proste

```
from django.shortcuts import redirect

def tajne_view(request):
    if not request.user.is_authenticated():
        return redirect('/login/?next=%s' % request.path)
```

Inne rozwiązanie

wersja z dekoratorem

```
from django.contrib.auth.decorators import login_required

@login_required
def tajne_view(request):
```

W razie braku autentykacji przekierowuje na stronę określoną w `settings.LOGIN_URL`.

Dodatkowe domyślne mechanizmy

- system grup użytkowników;
- system uprawnień.

Na koniec

Czego nie ma w standardowym django

- sprawdzanie siły hasła
- wsparcie dla innych systemów użytkowników;
- ograniczenie prób nieudanego logowania.

Plan wykładu

Hosting Django

- `www.heroku.com`: hobbistyczne projekty: free, obsługa Ruby, Java, Scala, Python, Django, ...
- `www.pythonanywhere.com`: od 0$/miesiąc;
- `www.webfaction.com`: Django, PHP, Perl, Rails, ...
- `https://www.linode.com/`: Linux na wirtualnej maszynie

Hosting Django

- `www.heroku.com`: hobbistyczne projekty: free, obsługa Ruby, Java, Scala, Python, Django, ...
- `www.pythonanywhere.com`: od 0$/miesiąc;
- `www.webfaction.com`: Django, PHP, Perl, Rails, ...
- `https://www.linode.com/`: Linux na wirtualnej maszynie

http://djangohosting.com/

Strona z odnośnikami do Djangowych hostingów z komentarzami.

Mechanizmy bezpieczeństwa

domyślny HTML escaping

Zamiana tagów html w szablonach:

< na `<`

> na `>`

' na `&#39;`

& na `&`

Użycie HTTPS zamiast HTTP

Trzymanie plików źródłowych poza katalogiem udostępnionym serwerowi WWW.

Cross site scripting (XSS)

Opis ataku

Osadzenie obcego kodu (np. JavaScript) na stronach WWW; przykładowo wklejając ten kod jako komentarz na forum.

Obrona

HTML escaping, ale nie obroni przed wszystkimi atakami.
Ostrożność przy zapamiętywaniu obcego html'u.

Cross site request forgery (CSRF)

Opis ataku

Nieświadome wykonanie akcji wymagającej uprawnień, np. poprzez kliknięcie na spreparowany link.

Obrona

Dodanie do formularzy ukrytego pola z losowym tokenem:

```
<form action="." method="post">{% csrf_token %}
```

i zabezpieczenie widoku

```
from django.views.decorators.csrf import csrf_protect
from django.shortcuts import render

@csrf_protect
def widok(request):
```

SQL injection

Opis ataku

Wstrzyknięcie obcego kodu do zapytania SQL

Obrona

Standardowo zapytania SQL są zabezpieczane poprzez dodanie ukośników przed znakami specjalnymi.

Clickjacking

Opis ataku

Oryginalna strona jest w sposób ukryty osadzana jako ramka na stronie na złośliwym serwerze.

Obrona

Zakazanie osadzania strony poprzez dodaniu taga `X-Frame-Options: deny` w odpowiedzi serwera:

```
X_FRAME_OPTIONS = 'DENY'
```