Django formos ir valdymo lygmuo

Albertas Gimbutas

[1]Matematikos ir informatikos institutas
Vilniaus universitetas

2014 pavasaris

Egzamino klausimų gairės

1. PEP8 kodavimo standarto taisyklės (0.3b)
2. Python programavimo kalbos savybės (iš Python tutorial) (1.2b)
   1. Kintamųjų vertimas į logines reikšmes (pvz.: `a = 0; bool(a)`).
   2. Funkcijų argumentų perdavimas su `*args` ir `**kwargs`.
   3. Paveldėjimas.
   4. Reguliariosios išraiškos.
   5. Kas yra iteratoriai ir generatoriai.
3. Bendras supratimas iš viso kurso (0.5b)
   1. Apibūdinkite modelis-vaizdas-kontroleris (MVC) šabloną? Už ką atsakinga kiekviena dalis?
   2. Kas yra migracijos, kam jos naudojamos?
   3. Kas yra pavyzdiniai objektai (angl. fixtures), kam jie naudojami?
   4. Kas yra įmitavimo objektas (angl. mock), kam jis naudojamas?
   5. Kokios testais grįsto programavimo geriosios ir blogosios savybės?
   6. Kas bendro tarp atviro kodo ir Python?

Turinys

1. Django formos
2. Funkcijomis grįsti kontroleriai
3. Klasėmis grįsti kontroleriai

Django formos

Django **formos** skirtos:

- Duomenų įvedimo lauko pavazdavimui.
- Įvestų duomenų tinkamumo patikrinimui.
- Validavimo klaidų pavaizdavimui, įvedus netinkamus duomenis.
- Įvestų duomenų konvertavimui į Python duomenų tipus.

```
from django import forms

class PersonForm(forms.Form):
    name = forms.CharField()
    age = forms.IntegerField()
    email = forms.EmailField()
    is_verified = forms.BooleanField()
```

Django modeliais grįstos formos

models.py

```
class Person(models.Model):
    name = models.CharField(max_lenght=80)
    age = models.IntegerField()
```

forms.py

```
class PersonForm(forms.Form):  # Paprasta forma
    name = forms.CharField()
    age = forms.IntegerField()
```

arba

```
class PersonForm(forms.ModelForm):  # Pagal modelį
    class Meta:                     # sudaryta forma
        model = Person
        fields = ['name', 'age']
```

Formos validavimas

views.py

```
def create_person_view(request):
    if request.method == 'POST':   # Data submitted?
        form = PersonForm(request.POST): # Fills form
        if form.is_valid():          # Is data valid?
            person = form.save()   # create/update object
            return HttpResponseRedirect(reverse('home'))
    else:
        form = PersonForm()   # Creates empty form
    return render(request, 'contact.html', {'form': form})
```

- `form.is_valid()` metodas patikrina ar atsiųstieji duomenys yra validūs:
  - Jeigu duomenys validūs, `form.is_valid()` užpildo `form.cleaned_data` žodyną su į Python tipus konvertuotais duomenimis.
  - Jeigu duomenys nevalidūs, `form.is_valid()` papildo formą validavimo klaidų pranešimais.

Formos pavaizdavimas Django šablone

```
<form action="." method="post">{% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Submit" />
</form>
```

- django: Cross Site Request Forgery protection

Kontrolerio grąžinami objektai

```
from django.http import Http404
from django.shortcuts import render, HttpResponse,
                                    HttpResponseRedirect

def my_view(request):
    # Grąžina Http atsakymą su HTML turiniu
    return HttpResponse(html)

    # Grąžina Http atsakymą: šabloną su užpildytu kontekstu
    context = {'name': 'Tom'}
    return render(request, template='index.html', context}

    # Grąžina Http atsakymą su nukreipimu į kitą puslapį
    return HttpResponseRedirect(url)

    # Sukelia klaidą, kurios dėka yra grąžinamas
    # Http atsakymas "Puslapis nerastas"
    raise Http404
```

Klasėmis grįsti kontroleriai

```
from django.views.generic import (CreateView, UpdateView,
                                  DeleteView)

class PersonCreate(CreateView):
    model = Person

class AuthorUpdate(UpdateView):
    model = Person
    form_class = PersonForm
    template_name = 'create_person.html'

class AuthorDelete(DeleteView):
    model = Person
    success_url = reverse_lazy('author-list')
```

- python.org: generic class-based views

Klasėmis grįstas objektų sąrašo kontroleris

views.py

```
class PersonList(ListView):
    model = Person
    paginate_by = 30
    success_url = reverse_lazy('topics')

    def get_context_data(self, **kwargs):
        context = super(PersonList, self).get_context_data(
        context['statistics'] = gen_statistics()
        return context
```

person_list.html

```
{% for person in object_list %}
    {{ person }}
{% endfor %}
```

- python.org: class-based views