Kurs rozszerzony języka Python

Środowisko Django — początki

Marcin Młotkowski

5 stycznia 2018

Plan wykładu

Plan wykładu

Co to jest

Django

Oparty o Pythona framework do tworzenia aplikacji internetowych.

Co to jest

Django

Oparty o Pythona framework do tworzenia aplikacji internetowych.

Wzorowany na *Ruby on Rails*.

Zamierzenie twórców (Adrian Holovaty, Simon Willison)

Framework webowy dla perfekcjonistów (z terminami)

Zalety

- wygodne definiowanie modeli, widoków, kontrolerów;
- czytelny podział kodu;
- wsparcie dla testowania;
- system cache;
- wbudowana autentykacja;
- nacisk na odporność na ataki.

Podstawowe pojęcia

Model

Reprezentacja określonego typu danych, na podstawie modelu konstruuje się schemat bazy danych czy klasę.

Podstawowe pojęcia

Model

Reprezentacja określonego typu danych, na podstawie modelu konstruuje się schemat bazy danych czy klasę.

View

Implementacja akcji na danych (logika biznesowa).

Podstawowe pojęcia

Model

Reprezentacja określonego typu danych, na podstawie modelu konstruuje się schemat bazy danych czy klasę.

View

Implementacja akcji na danych (logika biznesowa).

Template

Sposób prezentacji danych (modeli) a także interakcji z użytkownikiem.

Specyfika aplikacji WWW

Routing

Powiązanie żądań HTTP z odpowiednim kodem (funkcją bądź metodą).

Uwagi techniczne

Wersje

Najnowsza wersja: 2.0.1 (styczeń 2018). Współpracuje z Pythonem 2.7, 3.*

Projekt

Kolekcja różnych aplikacji wraz z konfiguracją (baza danych, konfiguracja www etc).

Aplikacja

Kod realizujący jakąś funkcjonalność.

Plan wykładu

Zadanie

System zapisów studentów na zajęcia.

Bardziej szczegółowy opis

Rodzaje danych

- Wykładowcy
- Studenci
- Zajęcia

Bardziej szczegółowy opis

Rodzaje danych

- Wykładowcy
- Studenci
- Zajęcia

Akcje

- Zapisywanie/wypisywanie się studentów na zajęcia;
- Administracja: dodawanie/usuwanie studentów i wykładowców

Na początek

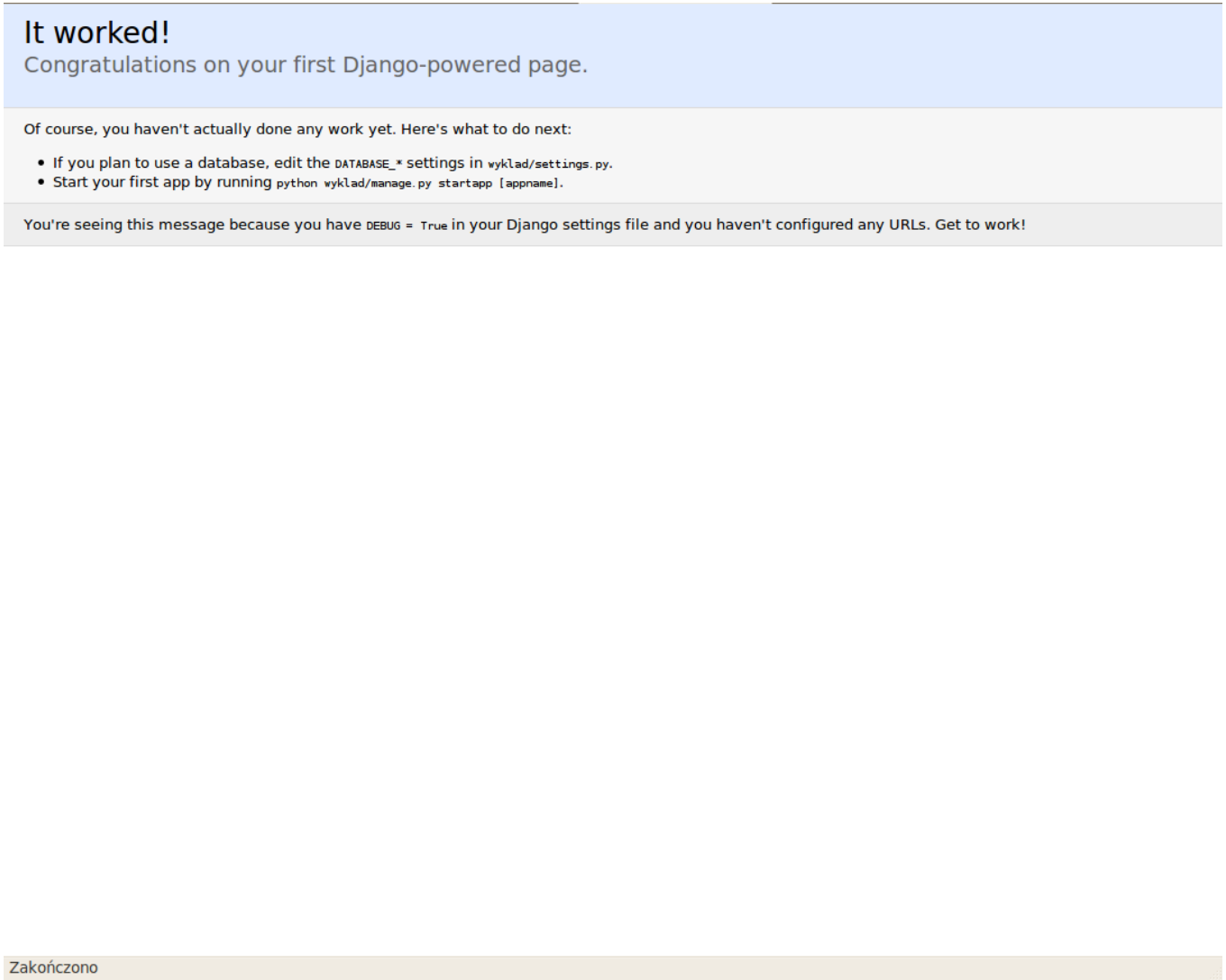
```
$ django-admin startproject wyklad
$ cd wyklad
$ python manage.py runserver
```

Na początek

```
$ django-admin startproject wyklad
$ cd wyklad
$ python manage.py runserver
```

```
Validating models...
0 errors found

Django version 1.1.1, using settings 'wyklad.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```


It worked!
Congratulations on your first Django-powered page.
Of course, you haven't actually done any work yet. Here's what to do next:
If you plan to use a database, edit the DATABASE_* settings in wyklad/settings.py.
Start your first app by running python wyklad/manage.py startapp [appname].
You're seeing this message because you have DEBUG = True in your Django settings file and you haven't configured any URLs. Get to work!
Zakończono

Wyjaśnienie

django-admin startproject wyklad

Tworzy katalog `wyklad` i tworzy tam szkielet projektu.

Wyjaśnienie

django-admin startproject wyklad

Tworzy katalog `wyklad` i tworzy tam szkielet projektu.

manage.py

To ważny plik służący do zarządzania projektem.

Przypomnienie

Template

Sposób prezentacji danych (modeli) a także interakcji z użytkownikiem.

Funkcje w modułach

hello_view.py

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("aKuKu")
```

Funkcje w modułach

hello_view.py

```
from django.http import HttpResponse
def hello(request):
    return HttpResponse("aKuKu")
```

urls.py – routing

```
import hello_view
urlpatterns = patterns("",
    ("^hello/$", hello_view.hello),
    ...
```

Uwagi

- `HttpResponse('aKuKu')` jako odpowiedź wysyła tylko tekst 'aKuKu', bez znaczników html;
- podany wzorzec pasuje tylko do `/hello` i do `/hello/`

Bardziej dynamiczne strony

```
def hello(request):
    now = datetime.datetime.now()
    html = '''<html><body>Witaj!''' 
    html += '''Dziś mamy %s.</body></html>''' % now
    return HttpResponse(html)
```

Prawdziwe Djangowe szablony

```
<html>
<head><title>Strona powitalna</title></head>
<body>
<h1>Powitanie </h1>
<p>Szanowna Pani/Szanowny Pan {{ person_name }},</p>
<p> Dziękujemy za odwiedzenie {{ company }}
w dniu {{ termin | date:"F j, Y" }}.</p>
</body>
</html>
```

Znaczniki, wyrażenia

{{ zmienna }}

Znaczniki, instrukcje

```
{% if wyrażenie %}
{% else %}
{% endif %}

{% for item in item_list %}
    {{ item }}
{% endfor %}
```

Filtry

Zmiana zmiennej przed wyświetleniem

{{ zmienna | filtr_1 | filtr_2 }}

Renderowanie szablonów

```
from django.template import Context, Template
```

Budowanie szablonów

```
szablon = Template("Mam na imię {{ name }}.")
```

Renderowanie szablonów

```
from django.template import Context, Template
```

Budowanie szablonów

```
szablon = Template("Mam na imię {{ name }}.")
```

Renderowanie – ustalenie kontekstu

```
kontekst = Context({ "name": "Pyton"})
szablon.render(kontekst)
```

u"Mam na imię Pyton"

Zmienne i kontekst, dodatki

```
szablon = Template(" {{ osoba.imie }} ma lat {{ osoba.wiek }}")
kontekst = Context({ "osoba": Osoba() })
```

Widoki a szablony

Przypomnienie

```
def hello(request):
    now = datetime.datetime.now()
    html = '''<html><body>Witaj!''' 
    html += '''Dziś mamy %s.</body></html>''' % now
    return HttpResponse(html)
```

Szablony i widoki – prosta wersja

```
def hello(request):
    now = datetime.datetime.now()
    t = '''<html><body>Witaj!'''
    t += '''Dziś mamy {{ teraz }}.</body></html>'''
    szablon = Template(t)
    html = szablon.render(Context({ "teraz": now}))
    return HttpResponse(html)
```

Szablony i widoki

System szablonów:

- definuje się katalog z szablonami w pliku `settings.py`
- szablony zwykle mają rozszerzenie `.html`

Użycie szablonu

Wersja długa

```
def hello(request):
    t = get_template("osoba.html")
    html = t.render(Context({ "osoba" : Persona() }))
    return HttpResponse(html)
```

Użycie szablonu

Wersja długa

```
def hello(request):
    t = get_template("osoba.html")
    html = t.render(Context({ "osoba" : Persona() }))
    return HttpResponse(html)
```

Wersja krótka

```
from django.shortcuts import render_to_response

def hello(request):
    return render_to_response("osoba.html",
                { "osoba" : Persona() })
```

Porządkowanie szablonów

Podkatalogi

Można szablony umieszczać w podkatalogach:

```
render_to_response("prezentacja/osoba.html",
                   { "osoba" : Persona() })
```

Porządkowanie szablonów

Podkatalogi

Można szablony umieszczać w podkatalogach:

```
render_to_response("prezentacja/osoba.html",
                   { "osoba" : Persona() })
```

Uwaga: podkatalogi dotyczą katalogu wskazanego w `settings.py`.

Składanie stron

```
<html>
<body>
{% include 'includes/header.html' %}
{% include "includes/nav.html" %}
<p>
Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua.
</p>
{% include "includes/footer.html" %}
</body>
</html>
```

Dziedziczenie szablonów

Cel dziedziczenia

- budowanie serwisów o podobnym wyglądzie;
- unikanie powtarzania kodu html'owego.

Szablon podstawowy

```
<html>
<head>
<title>{% block title %}{% endblock %}</title>
</head>
<body>
<h1>Wykład z Pythona</h1>
{% block content %}{% endblock %}
{% block footer %}
<hr>
<p>Dzięki że wpadłeś na mój wykład.</p>
{% endblock %}
</body>
</html>
```

Dziedziczenie szablonów

```
{% extends "'szablon.html"' %}

{% block title %}Wstęp do Django {% endblock %}

{% block content %}
<p>Dzisiaj będę ględził o Django</p>
{% endblock %}
```

Na samym początku

Utworzenie aplikacji

```
$ python manage.py startapp zapisy
```

Na samym początku

Utworzenie aplikacji

```
$ python manage.py startapp zapisy
```

Terminologia: projekt i aplikacja

Aplikacja – zaimplementowane funkcjonalności (modele, widoki etc.)
Projekt – aplikacja (lub aplikacje, tj. dodatkowe moduły) plus konfiguracja (szczegóły bazy danych, serwera WWW etc)

Efekt działania skryptu

Utworzenie szkieletu aplikacji w katalogu `zapisy`

- szablon modeli;
- szablon widoków;
- i jeszcze inne...

Deklaracja modeli: zapisy/models.py

```
from django.db import models

class Wykladowca(models.Model):
    imie = models.CharField(max_length=40)
    website = models.URLField()


class Student(models.Model):
    imie = models.CharField(max_length=40)

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

Weryfikacja poprawności modelu

W pliku settings.py

```
INSTALLED_APPS = ( "django.contrib.auth",
"django.contrib.contenttypes", "django.contrib.sessions",
"django.contrib.sites", 'wyklad.zapisy', )
```

Weryfikacja poprawności modelu

W pliku settings.py

```
INSTALLED_APPS = ( "django.contrib.auth",
"django.contrib.contenttypes", "django.contrib.sessions",
"django.contrib.sites", 'wyklad.zapisy', )
```

Konfiguracja bazy danych: settings.py

```
DATABASE_ENGINE = "sqlite3"
DATABASE_NAME = "./plik.db"
```

Weryfikacja poprawności modelu

W pliku settings.py

```
INSTALLED_APPS = ( "django.contrib.auth",
"django.contrib.contenttypes", "django.contrib.sessions",
"django.contrib.sites", 'wyklad.zapisy', )
```

Konfiguracja bazy danych: settings.py

```
DATABASE_ENGINE = "sqlite3"
DATABASE_NAME = "./plik.db"
```

$ python manage.py validate

Utworzenie struktury bazy danych

Kontrola utworzenia tabel

```
$ python manage.py sqlall zapisy
```

```
CREATE TABLE "zapisy_wykladowca" (
    "id" integer NOT NULL PRIMARY KEY,
...
CREATE TABLE "zapisy_wyklad" (
    ...
    "wykladowca_id" integer
          NOT NULL REFERENCES "zapisy_wykladowca" ("id")
```

Konfiguracja bazy danych

Utworzenie bazy danych

```
$ python manage.py syncdb
```

Konfiguracja bazy danych

Utworzenie bazy danych

```
$ python manage.py syncdb
```

Domyślnie (jeśli nie wykomentujemy modułów z settings.py) włączany jest system autentykacji i zakładane jest konto administratora.

Dostęp do danych

Porada

Można skorzystać z shella Djangowego

```
python manage.py shell
```

Tworzenie obiektów z modelu

```
from zajecia.models import Student
s1 = Student(imie="Ewa", website="")
s1.save()
s2 = Student(imie="Adam", website="")
s2.save()
lista_stud = Student.objects.all()
>>> [<Student: Student object>, <Student: Student object>]
```

Podpowiedzi

Podpowiedź 1: jednoczesne tworzenie i zapisywanie

```
s1 = Student.objects.create(imie="Ewa", website="")
```

Podpowiedzi

Podpowiedź 1: jednoczesne tworzenie i zapisywanie

```
s1 = Student.objects.create(imie="Ewa", website="")
```

Ładniejsze informacje

```
class Student(models.Model):
    ...
    def __unicode__(self):
        return self.name
```

Operacje na danych

Modyfikacja

```
p1.imie = "Ania"
p1.save()
```

Operacje na danych

Modyfikacja

```
p1.imie = "Ania"
p1.save()
```

Filtrowanie danych

```
lista = Student.objects.filter(imie="Ania")
```

Operacje na danych

Modyfikacja

```
p1.imie = "Ania"
p1.save()
```

Filtrowanie danych

```
lista = Student.objects.filter(imie="Ania")
```

Pobranie pojedynczego elementu

```
student = Student.objects.get(id=11)
```

Operacje na danych

Modyfikacja

```
p1.imie = "Ania"
p1.save()
```

Filtrowanie danych

```
lista = Student.objects.filter(imie="Ania")
```

Pobranie pojedynczego elementu

```
student = Student.objects.get(id=11)
```

Usuwanie

```
s1.delete()
Student.objects.all().delete()
```