Python OO programming

Het Django web framework

Kristof Michiels

Het Django web framework

Het Django web framework

- Website: https://djangoproject.com
- Django wordt omschreven als een "high-level, Python web framework met een helder, pragmatisch design dat snel ontwikkelen toelaat”
- Wat verstaan we onder "web framework"? Een verzameling tools die als pakket kunnen gebruikt worden om webtoepassingen te bouwen
- Gechreven in Python uiteraard ;-) Huidige versie: 4.1.1
- Documentatie: https://docs.djangoproject.com/en/4.1/
- Gebruik deze documentatie als leidraad!

Welke mogelijkheden biedt Django?

- ORM (object-relational mapper)
- URL-routing
- HTML templating (DTL)
- Verwerken van forms
- Unit testing
- Het is een platform om je eigen apps te ontwikkelen
- Gebruik als back-end, api voor apps in bvb React

Django installeren

- Binnen een virtual environment: pip install django

Een Django project aanmaken

Een Django project aanmaken

```
python -m django startproject naam_van_project
```

- manage.py: hiermee voer je Django-taken uit
- Folder "naam_van_project" wordt aangemaakt. Deze bevat op zijn beurt enkele bestanden:
  - __init__: geeft aan Python mee dat deze folder Python bestanden bevat
  - wsgi.py en asgi.py: hooks voor web servers Apache of nginx
  - settings.py: configureren van Django
  - urls.py: URL-routing

Django project opstarten

- Doe je via de terminal met: *python manage.py runserver*
- Zorg dat je je in de projectfolder bevindt
- Applicatie openen op http://127.0.0.1:8000

Django toepassingen

Een Django toepassing maken

- Een Django toepassing (of applicatie) mag je beschouwen als een component binnen een Django project
- Een project kan meerdere toepassingen bevatten
- Bvb een onderdeel van een groter geheel met een specifiek doel
- Zorg dat je je in de projectfolder bevindt
- Je maakt de applicatie aan in de terminal met: *python manage.py startapp mijn_toepassing*

Django toepassingen

- Er wordt een nieuwe folder "mijn_toepassing" aangemaakt
- Er wordt een db.sqlite-bestand aangemaakt
- apps.py: settings specifiek voor deze app - meestal geen editing nodig
- models.py: datalaag voor deze toepassing
- views.py: logic en control flow voor de toepassing
- tests.py: unit tests voor de toepassing
- admin.py: admin interface voor de toepassing (registratie models)
- migrations folder: bestanden die dienen voor het maken van aanpassingen aan de databank

Onze toepassing toevoegen aan settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mijn_toepassing',
]
```

Werken met Django models en admin

Werken met Django models en admin

- Django maakt gebruik van het MVC-architectuur patroon
- MVC = Model - View - Controller conceptuele structuur
- Componenten: url-patronen, views, models, templates
- Anders genoemd: url patronen, views, models, templates
  - urls.py: beslissen om naar welke view door te sturen
  - views.py: functies die logica/control flow bevatten
  - models.py: elke view kan gebruik maken van Django models, beschreven als klassen
  - elke view kan gebruik maken van templates (gebruikelijk in een folder templates ondergebracht)

Django models

- Bevinden zich dus in models.py in de app-folder
- Vertegenwoordigen de datalaag van een Django toepassing
- Ze beschrijven de structuur van het datamodel
- Ze laten ons toe de databank te bevragen
- Elk model is een klasse die erft van django.db.models.Model en definieert data-velden als klasse-attributen

Django datavelden

models.py:

```
from django.db import models


class Kunstwerk(models.Model):
    ROUTES = [('T', 'Topstukken'), ('L', 'Long Tail')]
    titel = models.CharField(max_length=100)
    registrator = models.CharField(max_length=100)
    soort = models.CharField(max_length=30)
    oorsprong = models.CharField(max_length=200, blank=True)
```

Django datavelden

- Tekst: CharField, TextField, EmailField, UrlField
- Numeriek: IntegerField, DecimalField
- Ander: BooleanField, DateTimeField
- Relationele data: ForeignKey, ManyToManyField
- Attributen: bvb max_length=10, blank=True, null=True, blank=True
- Model field reference: https://docs.djangoproject.com/en/4.1/ref/models/fields/

Django models en datavelden

```
from django.db import models

class Kunstwerk(models.Model):
    ROUTES = [('T', 'Topstukken'), ('L', 'Long Tail')]
    titel = models.CharField(max_length=100)
    registrator = models.CharField(max_length=100)
    soort = models.CharField(max_length=30)
    oorsprong = models.CharField(max_length=200, blank=True)
    beschrijving = models.TextField()
    route = models.CharField(max_length=1, choices=ROUTES, blank=True)
    aangemaakt_op = models.DateTimeField()
    jaar = models.IntegerField(null=True)
    kunstenaars = models.ManyToManyField('Kunstenaar', blank=True)

class Kunstenaar(models.Model):
    voornaam = models.CharField(max_length=100)
    naam = models.CharField(max_length=100)
```

Migraties

- Zoals reeds aangegeven: models beschrijven de structuur van het datamodel
- Migraties of migrations genereren scripts om de databank-structuur aan te passen aan dit datamodel
- Elke code-aanpassing aan het model leidt tot een nieuwe migratie
- Wanneer? Als je een model toevoegt, als je een dataveld toevoegt/wijzigt/verwijdert
- Migraties moeten worden aangemaakt en daarna uitgevoerd
- Zijn genummerde bestanden in folder migrations. Eerste die wordt aangemaakt: 0001_initial.py
- Django gebruikt standaard sqlite: DB Browser voor SQLite / http://sqlitebrowser.org
- Naamgeving: toepassing_klassenaam, met een autogegenereerd id-veld

Migraties

- aanmaken: *python manage.py makemigrations*
- tonen: *python manage.py showmigrations*
- migreren naar nieuwste toestand: *python manage.py migrate*
- migreren naar bepaald punt bvb: *python manage.py migrate mijn_toepassing 4*

Werken met Django admin

- Met Django admin creëren we een administratieve interface voor onze toepassing
- Het laat admin-gebruikers toe data te zien en te editen
- Bestand: admin.py

```
from django.contrib import admin

from .models import Kunstwerk

@admin.register(Kunstwerk)
class KunstwerkAdmin(admin.ModelAdmin):
list_display = ["naam", "soort", "oorsprong", "beschrijving", "route",
    "jaar", "kunstenaars"]
```

Werken met Django admin

- Eerst een superuser aanmaken: *python manage.py createsuperuser*
- Daarna terug server starten: *python manage.py runserver*
- Geeft je een automatische goeie minimalistische admin-interface
- __str__-method in model soms nodig om heldere naamgeving te krijgen

```
class Kunstenaar(models.Model):
    voornaam = models.CharField(max_length=100)
    naam = models.CharField(max_length=100)

    def __str__(self):
        return self.voornaam + " " + self.naam
```

Data bevragen met de Django ORM

- ORM = Object-relational mapper (herinner u SQL-Alchemy)
- Kan in de shell getest worden: *python manage.py shell*
- Geeft een QuerySet terug = collection van objecten
- Je kan gebruik maken van filters

```
from mijn_toepassing.models import Kunstwerk
Kunstwerk.objects.all()
```

Data bevragen met de Django ORM

```
kunstwerken = Kunstwerk.objects.all()
kunstwerk = kunstwerken[0]
kunstwerk.titel
kunstwerk.jaar
kunstwerk.beschrijving
kunstwerk.id
ander_kunstwerk = Kunstwerk.objects.get(id=1)
nog_ander_kunstwerk = Kunstwerk.objects.get(id=2)
onbestaand_kunstwerk = Kunstwerk.objects.get(id=9999) # geeft DoesNotExist-exception
Kunstwerk.objects.get(jaar=2034) # geeft MultipleObjectsReturned-exception
# gebruiken vooral filters om meerdere objecten terug te krijgen
kunstwerk.kunstenaars.all()
# QuerySet met kunstenaars dankzij de __str__-magic method
```

URL-routing en views

Werken met URL-patronen

- Deze patronen worden ook url-confs genoemd. We definiëren ze in urls.py
- Aan elk patroon wordt een view gekoppeld dat moet gebruikt worden indien patroon matcht
- Elk patroon wordt in volgorde nagekeken tot een match wordt gevonden
- Indien geen match op het einde wordt een 404-response teruggegeven
- Er wordt een path-functie gebruikt met 3 argumenten:
  - patroon of path converter: string gedeelte + "capture group" voor variabelen
  - View die moet worden gebruikt
  - Name: optioneel, gebruikt om dynamische links aan te maken

URL-patronen

- Documentatie: https://docs.djangoproject.com/en/4.1/topics/http/urls/

```
from mijn_toepassing import views
from andere_toepassing import views as andereviews  # eventueel

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', views.home, name='home'),
    path('kunstwerken/<int:kunstwerk_id>/', views.kunstwerk_detail,
        name='kunstwerk_detail')
]
```

Views aanmaken in views.py (zonder templates)

```
from django.shortcuts import render
from django.http import HttpResponse

def home(request):
    return HttpResponse("<p>home view</p>")

def kunstwerk_detail(request, kunstwerk_id):
    return HttpResponse(f"<p>kunstwerk_detail view met id {kunstwerk_id}</p>")
```

Views aanmaken in views.py (met templates)

- In templates folder kan je templates aanmaken

```
from django.shortcuts import render
from django.http import Http404

from .models import Kunstwerk


def home(request):
    kunstwerken = Kunstwerk.objects.all()
    return render(request, 'home.html', {
        'kunstwerken': kunstwerken,
    })
```

Views aanmaken in views.py (met templates)

```
def kunstwerk_detail(request, kunstwerk_id):
    try:
        kunstwerk = Kunstwerk.objects.get(id=kunstwerk_id)
    except Kunstwerk.DoesNotExist:
        raise Http404('Het kunstwerk werd niet gevonden')
    return render(request, 'kunstwerk_detail.html', {
        'kunstwerk': kunstwerk,
    })
```

Werken met Django templates

Django templates

- Zijn html-bestanden met extra syntax
- Django: DTL of optioneel, Jinja2
- Template tags: worden gebruikt voor loops, ifs enz...
- Template filters: nemen een string als input en geven een string als output (zoals pipe-symbool in shell scripting)
- conventie: zelfde naam als view-functie / geplaatst binnen folder templates in de toepassing-folder

```
{{ variabele }}
{% tag %}
{{ variabele|filter }}
```

Django templates

```
{{ kunstwerk.titel|capfirst}}

{% for kunstwerk in kunstwerken %}
{% endfor %}

{% url "home" %} => "/"
{% url "kunstwerk_detail" kunstwerk.id %} => "kunstwerken/1/"
```

Django templates (overerving)

base.html

```
{% block content %}
{% endblock content %}
```

home.html

```
{% extends "base.html" %}
{% block content %}
...
{% endblock content %}
```

Een template aanmaken (home.html)

```
<ul>
    {% for kunstwerk in kunstwerken %}
        <li>
            <a href="{% url 'kunstwerk_detail' kunstwerk.id %}">
                <h2>{{ kunstwerk.titel|capfirst }}</h2>
            </a>
            <p>{{ kunstwerk.soort }}</p>
            {% if kunsterk.oorsprong %}
                <p>Oorsprong: {{ kunstwerk.oorsprong }}</p>
            {% endif %}
            <p>{{ kunstwerk.beschrijving }}</p>
        </li>
    {% endfor %}
</ul>
```

Een template aanmaken (kunstwerk_detail.html)

```
<h1>{{ kunstwerk.titel|capfirst }}</h1>
<p>{{ kunstwerk.soort }}</p>
{% if kunstwerk.oorsprong %}
    <p>Oorsprong: {{ kunstwerk.oorsprong }}</p>
{% endif %}
{% if kunstwerk.kunstenaars.all %}
        {% for kunstenaar in kunstwerk.kunstenaars.all %}
            {{ kunstenaar.voornaam }} {{ kunstenaar.naam }}
        {% endfor %}
{% endif %}
<p>Aangemaakt door: {{ kunstwerk.registrator }}</p>
<p>Aangemaakt op: {{ kunstwerk.aangemaakt_op|date:"d M Y" }}</p>
<p>{{ kunstwerk.beschrijving }}</p>
```

Templates structureren

- base.html bevat de basis-html
- home.html: *{% extends "base.html" %}*

CSS en JS integreren met static assets

- folder static aanmaken in projectfolder
- Settings.py aanpassen:

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static')
]
```

CSS en JS integreren met static assets

- In de templates file (hier base.html):

```
{% load static %}
{% static 'style.css' %}
{% static 'images/logo.png' %}
{% static 'images/hoofding.jpg' %}
<script src="{% static 'main.js' %}"></script>
```

Python OO programming - les 3 -

kristof.michiels01@ap.be