

Python kontra PHP

czyli węże i słonie

Trend web 2.0 przyniósł ze sobą nie tylko nowe podejście do tworzenia stron internetowych, ale również nowe technologie i narzędzia. Wielu z nas słyszało lub miało kontakt z Ruby on Rails, a użytkownicy portalu grono.net korzystali z aplikacji napisanej w Django.

Dowiesz się...

- Zapoznasz się z nowymi narzędziami do tworzenia stron www.

Powinieneś wiedzieć...

- Podstawowa znajomość PHP i wzorca MVC.

Poziom trudności



W królestwo PHP zaczynają wkraczać inne języki oferując konkurencyjne rozwiązania, które to omówimy w tym artykule. Cytując wikipedię Python jest interpretowanym, interaktywnym językiem programowania stworzonym przez Guido van Rossum w 1990 roku. Rozwijany jest jako projekt *Open Source*, zarządzany przez niedochodową *Python Software Foundation*. PHP to wykonywany po stronie serwera skryptowy język programowania służący do generowania stron internetowych. Python tak samo jak *Ruby* sam z siebie nie jest przystosowany do tworzenia stron www, jako że jest to język uniwersalny od początku swojego istnienia. PHP natomiast od początku tworzony był jako język do tworzenia stron www. W niniejszym artykule skupimy się nie tyle na porównaniu języków, ale frameworków napisanych w Pythonie i PHP.

Różnice i podobieństwa

Oba języki mają wiele wspólnego, ponieważ są:

- Interpretowanymi językami wysokiego poziomu z dynamicznym systemem typów;
- Rozwijane jako projekty OpenSource;
- Posiadają duże społeczności;
- Łatwe w nauce (w porównaniu do np. Javy);
- Łatwe do rozszerzenia poprzez C/C++;

- Działają na wielu platformach i architekturach.

Mimo podobieństw są to dwa różne języki. Zbiór różnic przedstawia Tabela 1.

Nie ulega wątpliwości, że PHP jest popularny i ma bardzo dużą społeczność, również w Polsce. Znalezienie pomocy, czy to w postaci opublikowanych materiałów, czy też w postaci forum dyskusyjnego to nie problem. Również praktycznie każdy darmowy, tani hosting będzie zawierał w ofercie obsługę skryptów PHP.

Python ma mniejszą społeczność, w Polsce jest raczej mało popularny, lecz w skali światowej społeczność jest dość duża i skupiona wokół mniejszej ilości serwisów i organizacji. W kwestii hostingu umożliwiającego obsługę aplikacji *Zope*, *Django* czy *Pylons* nie będziemy mieć licznego grona kandydatów, lecz jakość usług będzie wysoka (np. *WebFaction*). Skrypty PHP wystarczy umieścić na serwerze, korzystając np. z klienta FTP. W przypadku aplikacji wspomnianych frameworków praktycznie nieodzowny będzie dostęp do *shella* oraz odpowiednia konfiguracja serwera www. Frameworki Pythonowe nie nadają się więc jako „Moja pierwsza strona”.

Kolejnym punktem, który warto poruszyć to wsteczna zgodność Pythona. Aplikacja pisana pod Pythonem 2.5 będzie w stanie działać pod Pythonem 2.3. Potencjalne zmiany w kodzie ograniczą się do zmian nazw modułów czy sposobu ich użytkowania (chyba że użyjemy „starszej” nazwy modułu). W przypadku PHP jest jeden problem. PHP5 udostępniło programistom wiele dobrych roz-

wiązań, które zostały wykorzystane w wielu projektach.

Niestety na większości kont hostingowych wciąż dominuje PHP4. Wydanie PHP6 jeszcze bardziej spotęguje to zjawisko, a programiści oprócz wykonania swojej pracy będą musieli co chwilę uwzględniać wersję PHP, pod którą ma działać ich aplikacja.

Python dla programisty PHP

Programista PHP dość szybko może połączyć się w konkurencyjnym języku, choć będzie musiał przyzwyczać się do pewnych różnic. Pierwszą jest brak średnika na końcu linii, drugą grupowanie kodu za pomocą wcięć. Pętla for w PHP wygląda znajomo:

```
<?PHP
for($i=1;$i <=5;$i++)
{
    print $i.'  


```

W Pythonie wyglądać może tak:

```
for i in range(1,6):
    print i
```

Nieco większą różnicą jest brak tablic znanych z PHP, a obecność list, tupli i dzienników. Pierwsze dwa typy uznać można za tablice numeryczne, a dzienniki za tablice asocjacyjne. Oto przykład:

```
dziennik = {'klucz': 'wartosc'}
print dziennik['klucz']
lista = ['element1', 'element2']
print lista[1]
print lista[0]
tupla = ('element tupli1', 'element
        tupli2')
for i in tupla:
    print i
```

Nowością będzie przestrzeń nazw i importowanie modułów. W przypadku PHP w skrypcie mamy dostęp do wszystkich funkcji i klas danej instalacji PHP. W przypadku Pythona by uzyskać dostęp np. do klas obsługujących pliki ZIP musimy zaimportować odpowiedni moduł:

```
import zipfile
z = zipfile.ZipFile("plik.zip", "r")
for plik in z.namelist():
    print 'Plik: ', plik,
    bajty = z.read(plik)
    print 'ma', len(bajty), 'bajtow'
```

Wydajność i skalowalność

Oba języki są interpretowane (do wykonania skryptu używa się interpretera), lecz PHP nie jest kompilowany do bajtkodu i zapisywany do wielokrotnego użytku. Każde żądanie wysłane do serwera spowoduje zinterpretowanie i wykonanie kodu w locie. W przypadku Pythona kompilacja do bajtkodu jest automatyczna, natomiast użytkownicy PHP muszą skorzystać z kompilatorów nie wchodzących w skład domyślnej dystrybucji PHP, takich jak *Xcache*, *eaccelerator*, *APC* czy produktów firmy *Zend*. W bezpośrednim starciu bez zabiegów optymalizacyjnych Python wygra, lecz odpowiednio skonfigurowane PHP nie będzie miało się czego wstydić.

Python i niektóre jego frameworki mogą działać wielowątkowo, co jest wydajniejsze od zwykłego trybu pracy PHP, jako że taki tryb pracy zużywa mniej zasobów. Również w przypadku *Django* czy *Pylons* możemy skorzystać z różnych sposobów hostowa-

nia aplikacji – dla *Apache* mamy `mod_python` i `mod_fcgid`, dla *Lighttpd*, *Cherokee* czy *Nginx* możemy wykorzystać protokół *FastCGI* lub *SCGI*.

Za chwilę poznamy bliżej framework *Django*. W jego przypadku dodanie keshowania dla całego projektu, to dodanie jednego wiersza w ustawieniach, a do keshowania możemy użyć m.in. *memcache* – wydajnego systemu bazującego na pamięci RAM.

Stosując Pythona do tworzenia dynamicznych stron www, na pewno nie stracimy na wydajności gotowych aplikacji, a nawet dość często nawet zyskamy.

Frameworki Pythona

W Pythonie znajdziemy kilka (a nie kilkadziesiąt i więcej) frameworków do tworzenia stron www, które można wykorzystać. Prym wiedzie *Django* i *Pylons*, a trzeci – *TurboGears* znajduje się obecnie w fazie przejściowej między starą i stabilną serią 1.X a przygotowywaną serią 2.X zgodną z *WSGI*. Dobrze znane od dawna *Zope* i *Plone* też ewoluują, lecz to *Pylons* i *Django* pod sztandarem nowoczesności i *web 2.0* atakują układ *LAMP*.

Django to framework dla „profesjonalistów z terminami”, jak głosi jego motto. Jest to dobrze zintegrowany zestaw własnych komponentów takich jak *ORM* wysokiego poziomu, system szablonów czy mapper adresów URL. *Pylons* to framework bazujący na integracji „najlepszych ze swojego rodzaju” rozwiązań.

Wykorzystuje on istniejące podzespoły jak również umożliwia łatwą ich wymianę, tak więc nie jesteśmy ograniczani do np. jedne-

go systemu szablonów. Krótkie porównanie przedstawia Tabela 2.

Oba frameworki są bardzo dobrymi rozwiązaniami. *Django* jest frameworkiem najlepiej pasującym do aplikacji typu CMS. *Pylons* ma podejście „zrób to sam” i króluje w przypadkach, gdy wymagamy elastyczności i stawiamy nietypowe wymagania.

Portale takie jak *Grono.net* i *lawrence.com* wykonane są w *Django* – i są dużymi serwisami internetowymi. *Poradnikzdrowie.pl* to serwis napisany w *Pylons* zintegrowany z *RedDotem* (istniejącą wcześniej strukturą serwisów właściciela). Serwisy te dobrze obrazują różnice między *Django* a *Pylons*.

WSGI

WSGI czyli *Web Server Gateway Interface* to standard interfejsu aplikacji napisanej w Pythonie do porozumiewania się z serwerami. Stworzono standard interfejsu, który obecnie stosowany jest przez wiele pythonowych aplikacji, głównie frameworków i aplikacji sieciowych.

Przestrzeganie standardu ułatwiło współpracę wielu aplikacji ze sobą i z serwerami – pojawiło się pojęcie *middleware*, czyli skryptu o określonym interfejsie, modyfikującym żądanie bądź odpowiedź wysyłaną do lub z serwera.

Dany skrypt *middleware* może być bez modyfikacji stosowany w każdym frameworku, zgodnym z wytycznymi *WSGI*, co stanowi ogromne pole do rozbudowy narzędzi tego typu o nowe możliwości. Przykładem jest tu framework *Pylons* stawiający na *WSGI*. Sam nie posiada ani systemu użytkowników, ani

Tabela 1. Różnice pomiędzy językami PHP i Python

PHP	Python
Składnia bazująca na C i Perlu	Wcięcia stosowane do grupowania kodu
Instrukcje <i>switch</i> i <i>do... while</i>	Przestrzenie nazw
Brak przestrzeni nazw	Wszystko jest referencją
Nawiasy klamrowe stosowane do grupowania kodu	Wielowątkowość
Większa społeczność i popularność	Dużo typów wysokiego poziomu
Język głównie do tworzenia dynamicznych stron	Język ogólnego użytku
Stabilny, lecz gwałtowniej rozwijany (PHP4 – PHP5)	Wstecznie zgodny, stabilny i dojrzały. Standaryzacja rozwiązań
Kompilatory dostępne jako rozszerzenia (<i>APC</i> , <i>Xcache</i> , <i>eaccelerator</i> , <i>zend</i>)	Automatyczne kompilowanie do bajtkodu

Tabela 2. Porównanie frameworków Pythona

Django	Pylons
Dobrze zintegrowane własne komponenty	Integruje najlepsze istniejące komponenty i umożliwia ich łatwą wymianę
Łatwy w nauce, świetnie udokumentowany	Niespójna i mniej liczna dokumentacja (każdy komponent ma dokumentację na swojej stronie). Wymaga więcej czasu na naukę i konfigurację komponentów
ORM wysokiego poziomu obsługujący MySQL, SQLite i PostgreSQL (Oracle od wersji 1.0)	Zalecane stosowanie ORMa <i>SQLAlchemy</i> – większe możliwości i łatwiejsza praca na istniejących tabelach, lecz trudniejszy w nauce.
Większa społeczność, zastosowany w wielu projektach	Aplikacja może działać wielowątkowo
Generyczne widoki, Automatyczny panel admina, system użytkowników i uprawnień	Wiele pomocników JavaScript, Ajax (w tym <i>Scriptaculous</i>)

obsługi *openID*, ale middleware, takie jak np. *AuthKit*, bezproblemowo dodają te funkcjonalności do frameworka. Chęć przestrzegania specyfikacji jest w społeczności tak duża, że trzeci duży framework – *TurboGears* zawiesił rozwój dotychczasowego niezgodnego z WSGI drzewa 1.X tworząc zarazem rozwojową obecnie gałąź 2.X, zgodną z WSGI. Dla programistów rozwijających frameworka oznacza to przepisanie wielu modułów *TurboGears*.

W PHP standaryzacja rozwiązań kuleje. Nawet na poziomie nazw funkcji trafimy na różne style nazewnictwa. W kategorii komponentów sprawę nieco wyjaśnia *PEAR*, lecz czy *Codelgniter*, *Symfony* lub *Zend Framework* bezboleśnie integrują API pakietów *PEAR*? Niestety nie.

Zend Framework i *ezComponents* jako zbiór luźniejszych komponentów, które łatwiej wykorzystywać w innych frameworkach, dają nadzieje, lecz na tak daleko posuniętą standaryzację jak w przypadku Pythona nie mamy na razie co liczyć.

Django kontra Codelgniter i spółka

Zajmiemy się teraz porównaniem obu frameworków, jak i drogi tworzenia aplikacji z ich użyciem (*Codelgniter opisany został w numerze 2/2007 phpSolutions*). *Django* wykorzystuje wzorzec MVC, lecz w porównaniu z CI wprowadza kilka zmian. Pierwszą z nich jest nazwa – MTV czyli *Model-Template-View*. W *Django* model określa strukturę tabeli (logikę bazodanową ogólnie), widok zawiera logikę aplikacji, a szablon wygląd. Kolejna różnica dotyczy samego modelu. W przypadku *Codelgnitera* model zawierał definicje różnych operacji na bazie danych, takich jak pobieranie ostatnich newsów, kasowanie i aktualizowanie wpisów. W *Django* model definiuje strukturę tabeli, a operacje wykonujemy w widokach za pomocą *ORMa*.

Przykładowy projekt Django

Zaprezentuję teraz prosty projekt *Django* – blog z systemem newsów obrazujący sposób tworzenia projektów *Django*.

Listing 1. Podłączenie bazy danych do naszego CMSa

```
DATABASE_ENGINE =
    'sqlite3' # 'postgres', 'mysql', 'sqlite3' lub 'ado_mssql'.
DATABASE_NAME =
    'bazka.db' # nazwa bazy danych lub ścieżka dla bazy sqlite3
DATABASE_USER =
    '' # użytkownik bazy, puste dla sqlite3
DATABASE_PASSWORD =
    '' # hasło użytkownika, puste dla sqlite3
DATABASE_HOST =
    '' # host do bazy danych, puste localhost
DATABASE_PORT =
    '' # Podaj port jeżeli niestandardowy, puste dla sqlite3.
następnie zmieniamy język i strefę czasową:
TIME_ZONE = 'Europe/Warsaw'
LANGUAGE_CODE = 'pl'
```

Listing 2. Model aplikacji wiadomości

```
from django.db import models
class News(models.Model):
    news_title = models.CharField(
        maxlength=255, verbose_name='Tytuł')
    news_text = models.TextField(
        verbose_name='Treść')
    news_date = models.DateTimeField(
        auto_now_add = True, blank=
            True, verbose_name='Data dodania')
class Meta:
    verbose_name = "Wiadomość"
    verbose_name_plural = "Wiadomości"
class Admin:
    list_display = ('news_title', 'news_date')
    list_filter = ['news_date']
    search_fields = ['news_title', 'news_text']
    date_hierarchy = 'news_date'
def __str__(self):
    return self.news_title
```

Instalacja

Ostatnią wydaną wersją *Django* jest wersja 0.96. Pobieramy archiwum z frameworkiem ze strony projektu i rozpakowujemy. Otwieramy konsolę i przechodzimy do katalogu z kodem *Django*. Wydajemy polecenie:

```
python setup.py install
```

Django wymaga kilku dodatkowych bibliotek Pythona:

- Do obsługi *SQLite Django* używa *pysqlite*;
- *Postgres: psycopg*;
- *MySQL: mysql-python*.

Tworzenie aplikacji w Django

Tworzenie projektu. Projekt składa się z aplikacji, np. CMS z różnych modułów. By stworzyć projekt wystarczy wydać polecenie:

```
django-admin.py startproject NAZWA_PROJEKTU
django-admin.py startproject blog
```

Stworzone zostaną podstawowe pliki projektu, możemy nawet uruchomić serwer *Django* dla tego projektu.

Uruchamianie serwera Django

Z konsoli po przejściu do katalogu projektu wykonaj polecenie:

```
python manage.py runserver 8080
```

Komenda ta uruchomi deweloperski serwer *Django*. Będzie on dostępny pod adresem *http://localhost:8080/*.

Konfiguracja projektu – bazy danych

Django jest silnie powiązane z bazami danych i do dalszych operacji będzie nam potrzebna dostępna baza danych (*sqlite3*, *mysql* lub *postgres*). Edytujemy *settings.py* i ustawiamy *SQLite* jako naszą bazę danych: patrz Listing 1.

Teraz zatrzymujemy serwer *Django* i synchronizujemy bazę danych. Po dodaniu nowej aplikacji, bądź danych do bazy danych nowego projektu, musimy te dane zsynchronizować. Służą do tego polecenie:

```
python manage.py syncdb
```

W przypadku rozpoczynania pracy z bazą danych *Django* zapyta nas też o dane głównego admina projektu.

Tworzenie aplikacji

By stworzyć aplikację wystarczy wykonać polecenie z katalogu projektu:

```
python manage.py startapp
    NAZWA_APLIKACJI
python manage.py startapp news
```

Co stworzy katalog aplikacji wraz z plikami:

```
news/
  __init__.py
  models.py
  views.py
```

Zbiór *models.py* zawiera modele naszej aplikacji. Kod modelu dla naszej przykładowej aplikacji przedstawia Listing 2.

Każdy model, klasa dziedziczy z `django.db.models.Model`, przez co możemy zdefiniować strukturę poszczególnych tabel. Pole *CharField* i *TextField* służą do przechowywania tekstu, *DateTimeField* daty. Teraz edytujemy *settings.py* projektu. Do `INSTALLED_APPS` dodajemy naszą aplikację (`NAZWA_PROJEKTU.NAZWA_APLIKACJI`), czyli *blog.news*. Zapisujemy i synchronizujemy bazę danych.

Panel Admina

By *włączyć* panel administracyjny wykonujemy:

- Do `INSTALLED_APPS` w *settings.py* dodajemy `django.contrib.admin`;
- Synchronizujemy bazę danych;
- Edytujemy *urls.py* i usuwamy komentarz (#) z wiersza:


```
(r'^admin/', include('django.contrib.admin.urls'))
```

Panel Admina dostępny jest pod adresem `http://localhost:8080/admin/` i umożliwi zarządzanie aplikacją – dodawanie, usuwanie i edytowanie wpisów wraz z opcjami pomocniczymi jak listowanie i wyszukiwanie.

Interfejs użytkownika

Mamy gotowy panel administratora wiadomości, lecz nie mamy jeszcze interfejsu od strony użytkownika. Zaczynamy od zaprojektowania adresów URL. *urls.py* zawierają listę powiązanych adresów URL z akcjami wywoływanymi w przypadku dopasowania URLa. Format:

```
(wyrażenie regularne, funkcja pythona
 do wykonania [, opcjonalnie katalog])
```

urls.py powinien wyglądać tak: Listing 3.

W wielu frameworkach MVC, w tym miejscu, musielibyśmy napisać kontrolery (widoki w *Django*) wykonujące określone czynności. Lecz *Django* posiada coś takiego, jak *Generyczne Widoki*, czyli predefiniowane akcje takie jak: pokaż określony wpis, listuj wpisy ze stronicowaniem i inne. W powyższym kodzie *urls.py* użyliśmy generycznego `django.views.generic.list_detail.object_list` – listy wpisów ze stronicowaniem. Teraz musimy tylko stworzyć szablony.

Szablony

- W katalogu projektu utwórz katalog `templates` (na szablony) i `site_media` (na pliki statyczne);
- Dodaj `'templates'` (nazwę katalogu na szablony) w *settings.py* do `TEMPLATE_DIRS`.


```
TEMPLATE_DIRS = (
    'templates',
)
```

Edytuj plik *urls.py* i dodaj regułę:

```
(r'^site_media/(.*)$', 'django.views.static
    .serve',
 {'document_root': '/ŚCIEŻKA/DO/site_
    media'}),
```

Teraz wystarczy, że stworzymy szablony. Skorzystamy z darmowego szablonu. Założymy, że zawiera on XHTML zawierający katalog `images` oraz pliki: `default.css` i `index.html`.

- Katalog i plik `css` kopiujemy do `/site_media` a `index.html` do `/templates`;
- Edytujemy plik `index.html` i zastępujemy przykładową treść tagiem:


```
{% block content %}{% endblock %};
```
- Znajdujemy `<link rel="stylesheet" type="text/css" href="default.css" />`;
- Zamieniamy na `<link rel="stylesheet" type="text/css" href="/site_media/default.css" />`;
- Tworzymy plik `/templates/news_list.html` z kodem: Listing 4.

Listing 3. Dopasowanie adresów URL w *urls.py*

```
from django.conf.urls.defaults import *
from blog.news.models import *
urlpatterns = patterns('',
    (r'^admin/', include(
        'django.contrib.admin.urls')),
    (r'^/?$', 'django.views.generic.
        list_detail.object_list', {
        'queryset': News.objects.all().
        order_by('-id'), 'paginate_by': 10,
        'allow_empty': True, 'template_name':
        'news_list.html'}),
    (r'^(?P<page>[0-9]+)$',
        'django.views.generic.list_detail.object_list',
        {'queryset': News.objects.all().order_by('-id'),
        'paginate_by': 10, 'allow_empty': True,
        'template_name': 'news_list.html'}),
)
```

Listing 4. Przykładowy plik *news_list.html*

```
{% extends "index.html" %}
{% block content %}
    {% if object_list %}
        {% for new in object_list %}
            <h3>{{ new.news_title }}</h3>
            <p>{{ new.news_text }}... {{
            new.news_date|truncatewords:"1" }}</p>
        {% endfor %}

        {% if has_previous %}
            <div style="text-align:center;">
                <a href="/?page={{ previous }}">
                <b>Nowsze Wiadomości</b></a></div>
        {% endif %}
        {% if has_next %}
            <div style="text-align:center;">
                <a href="/?page={{ next }}">
                <b>Starsze Wiadomości</b></a></div>
        {% endif %}
        {% else %}
            Brak wiadomości
        {% endif %}
    {% endblock %}
```

Otwieramy główną stronę swojej aplikacji w przeglądarce, wpisując adres `http://localhost:8080/`. Efekt przedstawiono na Rysunku 2.

Trochę wyjaśniń, otóż zaczęliśmy od podania w `settings.py` nazwy katalogu na szablony, następnie w `urls.py` podaliśmy regułę obsługującą pliki statyczne pod serwerem deweloperskim (w warunkach produkcyjnych to serwer `www` tym się zajmuje). Pliki statyczne to grafiki, pliki `JS` czy `css`, jak i inne, do których chcemy mieć dostęp z poziomu aplikacji `Django`. Kolejną czynnością było przystosowanie szablonu HTML. Przypomnijmy sobie regułę serwującą statyczne pliki:

```
(r'^site_media/(.*)$', 'django.views.static
    .serve',
{'document_root': '/ŚCIEŻKA/DO/site_
    media'}),
```

`r'^site_media/(.*)$'` – oznacza, iż wszystkie pliki statyczne dostępne są przez `/site_media/plik_statyczny`, tak więc musieliśmy zmienić odnośnik do pliku `CSS`. Kolejnym etapem było zastąpienie przykładowej treści przez `tag bloku`. Następnie stworzyliśmy drugi szablon `dziedziczący index.html`:

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

Szablony `Django` obsługują dziedziczenie – `news_list.html` dziedziczy `index.html`, czyli wy-

świetla zawartość tego szablonu. Oprócz dziedziczenia skorzystaliśmy z drugiego ważnego elementu – bloków. W `news_list.html` podaliśmy zawartość dla bloku `content`:

```
{% block content %}
tutaj zawartość
{% endblock %}
```

W efekcie wypełniliśmy `index.html` określoną przez nas zawartością – listą newsów. Oczywiście możemy stworzyć identyczną aplikację w PHP, lecz kluczowym czynnikiem jest tutaj szybkość tworzenia. `Django` jest bardzo dobrym narzędziem, gdyż jego komponenty umożliwiają bardzo wydajne tworzenie wysokiej jakości aplikacji. W porównaniu do takiego samego programu wykonanego w `CodeIgniterze` różnice są następujące:

- Nie tworzymy tabeli w bazie danych, robi to framework (dbając m.in. o indeksy);
- Mamy gotowy `Panel Admina` do zarządzania aplikacją;
- `Generyczne Widoki` znacznie ułatwiają tworzenie interfejsu od strony użytkownika;
- Model opisuje tabele bazy danych, a nie operacje na nich – w przypadku `CI` bez struktury tabel trudno byłoby je odtworzyć bazując na kodzie modelu (brak wycieku informacji poza projekt);
- Walidacja na poziomie mapowania URLi poprzez wyrażenia regularne i na pozio-

mie modelu poprzez wysokopoziomowe typy pól.

Smaczki Django

Prezentowana przed chwilą aplikacja, nie pokazała wszystkich możliwości frameworka. Poznanie ich wymaga bardziej dokładnego poznania tego narzędzia, lecz poniżej opiszę kilka innych przydatnych komponentów `Django`.

ORM Django

ORM mapuje strukturę bazy danych – poszczególne tabeli na obiekty. W `Django` obsługa ORMa jest bardzo prosta, a sam komponent jest mocno rozbudowany. `Propele` dostępny dla PHP niestety nie może pochwalić się tak czystym i przyjemnym interfejsem.

Listing 5. przedstawia model dla systemu blogów złożony z trzech tabel. `Blog` – nazwa i opis blogów, `Author` – Nazwisko i email autorów. `Entry` – wpis dla danego bloga.

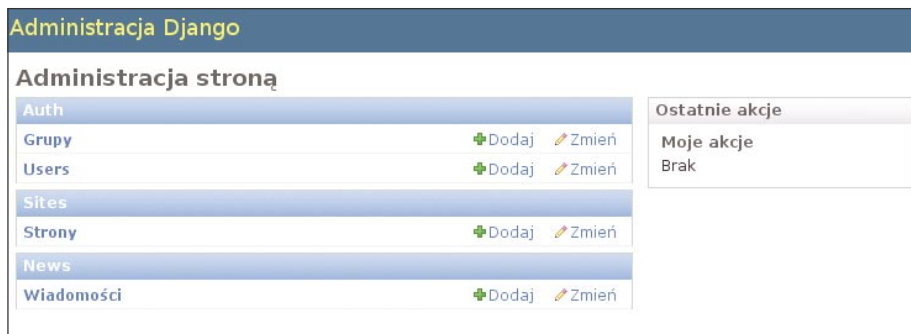
Zaprezentowany na listingu model prezentuje obsługę powiązanych tabel – `Entry` ma pole `authors` zależne od `Author` i pole `blog` zależne od `Blogs`. Proste zależności wiele-do-jednego określane są przez pola typu `ForeignKey`, zawierające jako parametr nazwę modelu. Tak więc dany wpis (`Entry`) przypisany jest do konkretnego autora i bloga. `Django` oferuje również obsługę zależności wiele-do-wielu czyli np. gdyby taka zależność dotyczyła pola `blog`, to dany wpis (`Entry`) mógłby być przypisany do wielu blogów. Listing 6. przedstawia operacje na rekordach wykonywane poprzez ORM.

System uprawnień i użytkowników

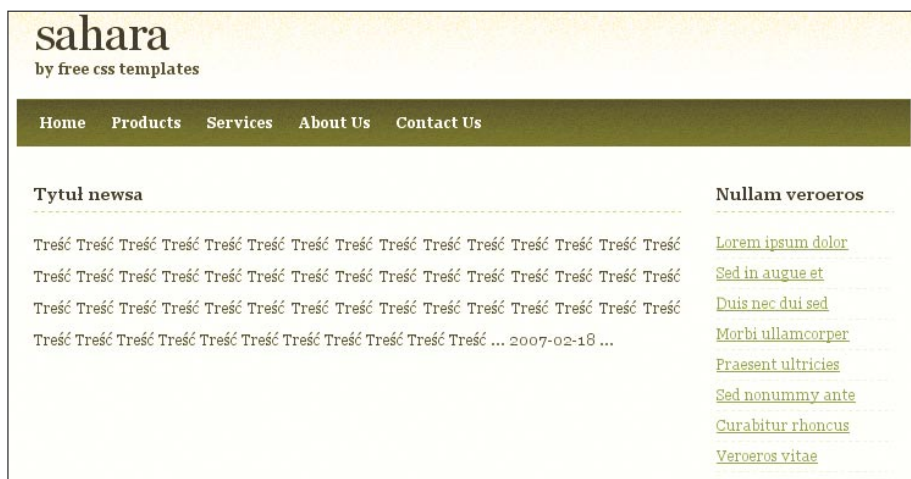
`Django` oferuje również system użytkowników, grup oraz uprawnień. Przy synchronizacji bazy danych mogłeś zauważyć, że dla każdego modelu (klasy) tworzone są domyślnie trzy „znaczniki” uprawnień: dodawanie, edycja i usuwanie obiektu danego modelu. Można też tworzyć własne. W Panelu administracji w łatwy sposób możemy tworzyć grupy, zbiory uprawnień jak również nadawać poszczególnym użytkownikom uprawnienia i przypisywać ich do istniejących grup. Wykorzystanie tego systemu w praktyce jest bardzo łatwe, jako że w widoku bieżący użytkownik reprezentowany jest przez obiekt `request.user`. Najlepiej obrazuje to Listing 7. zawierający kod prostego widoku.

Middleware

`Middleware` to pojęcie związane z WSGI, obecnym również w `Django`. `Middleware` może globalnie wpływać na żądania wysyłane do widoków, jak i na ich odpowiedzi. Do czego można zastosować `middleware`? Np. do śledzenia aktywności użytkowników na stronie – każde żądanie dowolnej strony



Rysunek 1. Panel Admin w akcji



Rysunek 2. Widok strony głównej wiadomości

naszej aplikacji może wykonać kod *middleware*, aktualizujący listę użytkowników *Na Stronie*. Może też posłużyć do modyfikacji nagłówków, czy zmiany działania serwisu w określonych warunkach (np. obsługa *openid* w *Pylons*).

Mapper adresów URL

Jak już mogliśmy zauważyć w *Django* (w *Pylons* również) adresy URL są mapowane, co w PHP jest niespotykane. Za pomocą wyrażeń regularnych określamy URL i przypisujemy mu widok, jaki ma zostać wykonany. URL, które nie zostaną dopasowane do żadnego wyrażenia nie istnieją (zwrócony zostanie kod 404). Wymaga to od programisty znajomości podstaw wyrażeń regularnych. W efekcie dostajemy przyjazne wyszukiwarkom odnośniki oraz w przypadku zastosowania poprawnych formuł – walidację na poziomie mapowania odnośników (nie trzeba tego robić w widoku). Mapowane adresy URL to nie przepisany *Query String*, który również istnieje i może być stosowany w wyjątkowych okolicznościach (zaleca się stosowanie mapowanych odnośników).

Zedo.pl – Dlaczego Django?

Zedo.pl to nowo powstały portal społecznościowy, w którym użytkownicy dzielą się komponentami do telefonów komórkowych, takimi jak tapety i w przyszłości motywy czy dzwonki. Pierwotnie właściciel chciał, by serwis był wykonany w znanej mu technologii – PHP i MySQL, lecz gdy zapoznałem go z *Django* i jego możliwościami zgodził się na to rozwiązanie. Dlaczego wybrałem *Django*? Ze względu na szybkość i przyjemność tworzenia kodu aplikacji. Główne problemy, jakie dostrzegłem po zapoznaniu się ze specyfikacją portalu to:

- Rozbudowany system użytkowników wraz z „użytkownikami online”, rozbudowanymi profilami z komentarzami, przyjaciółmi itp.;
- Walidacja formularzy, w szczególności dodawanie tapet (walidacja i skalowanie plików);
- Powiązane między sobą tabele (Komórki z przypisaną rozdzielczością – tapeta o określonej rozdzielczości) i filtrowanie danych z powiązanych tabel.

Nie jest to nic, czego nie można zrobić w PHP, ale tutaj liczy się czas programisty i wysiłek jaki należy włożyć, by uzyskać założony efekt. *Django* zawiera dobry system użytkowników i uprawnień oraz rozbudowany, elastyczny system walidacji i obsługi formularzy, dzięki temu dość łatwo rozwiązać możliwe problemy:

- Skrypt *Middleware* w przypadku zalogowanego użytkownika, co 5 minut aktualizuje jego obecność na stronie, wykorzystując *cookie* z zapisanym znacznikiem czasu, do określania czy i kiedy zaktualizować wpis w bazie danych;
- Do walidacji wysyłanych tapet, oprócz walidacji rozszerzenia wykorzystany został PIL do walidacji rozmiaru i typu MIME (wykrywa łączone pliki typu grafika+PHP);
- System użytkowników został rozszerzony o zależne tabele zapewniające funkcjonalność profilu użytkownika.

Nietypowe rozwiązania

Python to język ogólnego przeznaczenia, istnieje więc wiele komponentów, które nie były two-

R E K L A M A

.psd

Photoshop
Solutions
for Designers

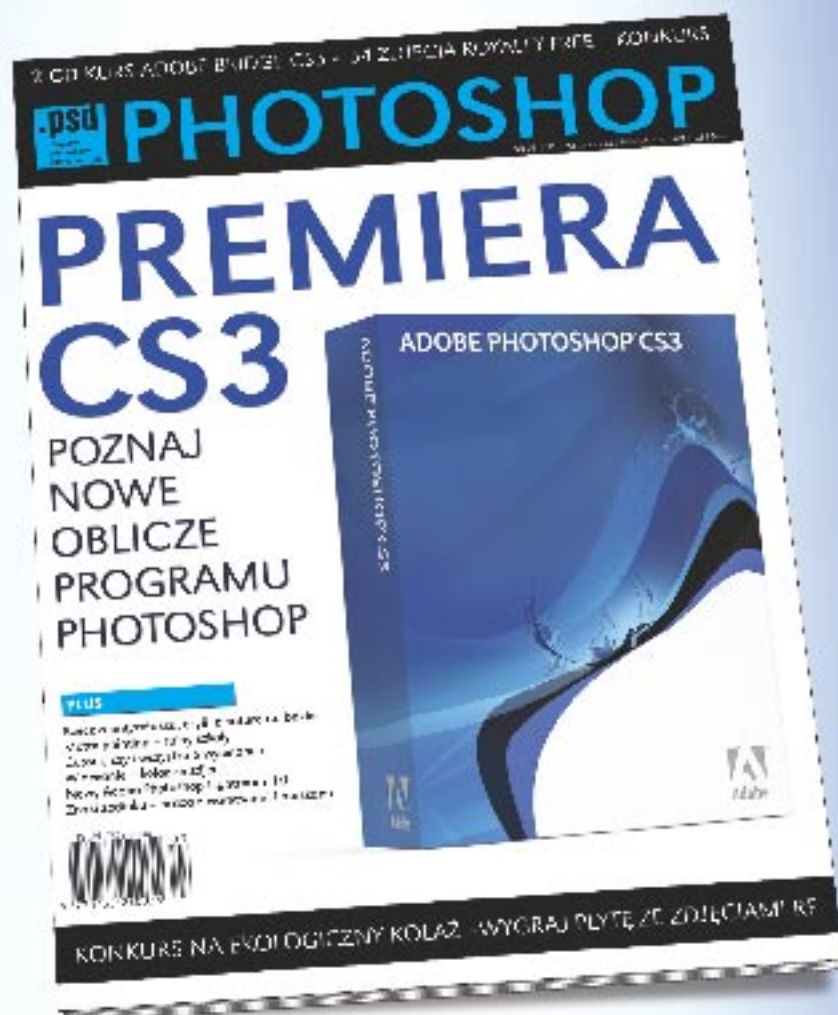
W każdym numerze:

- pliki źródłowe do tutoriali
- filmy instruktażowe
- multimedialny kurs Photoshopa

Ponadto zdjęcia Royalty Free
do wykorzystania komercyjnego!!!

**Multimedialny kurs
Adobe Bridge CS3**

www.psdmag.org



W Sieci:

- <http://www.python.org> – strona Pythona;
- http://www.ipersec.com/index.php?q=en/bench_ea_vs_apc – porównanie wydajności różnych enkoderów bajtkodu PHP;
- <http://www.python.org/dev/peps/pep-0333/> – opis WSGI;
- <http://wiki.rubyonrails.com/rails/pages/Framework+Performance> – porównanie wydajności Symphony, Ruby on Rails i Django;
- <http://twistedmatrix.com/trac/> – strona frameworka Twisted;
- <http://www.djangoproject.com> – strona Django;
- <http://www.pylonshq.com> – strona Pylons;
- <http://www.python.rk.edu.pl/> – polska dokumentacja Django i Pylons
- <http://docs.pythonweb.org/> – wiki Pylons

Listing 5. Model systemu blogów

```
class Blog(models.Model):
    name = models.CharField(maxlength=100)
    tagline = models.TextField()
    def __str__(self):
        return self.name

class Author(models.Model):
    name = models.CharField(maxlength=50)
    email = models.URLField()
    def __str__(self):
        return self.name

class Entry(models.Model):
    blog = models.ForeignKey(Blog)
    headline = models.CharField(maxlength=255)
    body_text = models.TextField()
    pub_date = models.DateTimeField()
    authors = models.ManyToManyField(Author)
    def __str__(self):
        return self.headline
```

Listing 6. Przykładowe operacje ORMa django

```
# Dodanie wpisu
b = Blog(name= 'Blog Andrzeja', tagline=
    'Najnowsze wieści z pola')
b.save()
#edycja
b.name='Blog Zbyszka'
b.save()
# pobranie wszystkich blogów
blogi = Blog.objects.all()
# pobranie 5 blogów
blogi = Blog.objects.all()[:5]
# filtrowanie i sortowanie wyników
wpisy = Entry.objects.filter(
    pub_date__year=2005).order_by(
    '-pub_date', 'headline')
```

Listing 7. Przykładowy widok wykorzystujący system użytkowników i uprawnień

```
def moj_widok(request):
    if request.user.is_authenticated():
        # zalogowany
    else:
        # niezalogowany
    if request.user.is_staff and
        request.user.has_perm(
            'news.add_news'):
        # zalogowany,
        z odpowiednimi uprawnieniamibv
```

rzony z myślą o stronach www, co nie oznacza, że nie możemy ich wykorzystać. Chcąc zrobić bazę danych wyników analiz chromatograficznych – wystarczy skorzystać z biblioteki pyCDF i zapisać wyniki w formacie CDF. Jeżeli chcemy generować złożone wykresy z tych analiz, lub z innych danych, problem nasz rozwiąże *matplotlib*. Generowaniem raportów, plików PDF czy prostych wykresów zajmie się *reportlab*, a skanowaniem przesyłanych załączników, pod kątem obecności wirusów, *pyClamAV*. Odpowiednikiem GD z PHP będzie PIL o równie bogatej, jeżeli nie większej funkcjonalności.

Programowanie sieciowe

Python w swojej bibliotece standardowej posiada szereg modułów przydatnych przy programowaniu sieciowym. Moduł *socket*, jak sama nazwa wskazuje, daje nam możliwość operowania na gniazdach. Moduły wyższego poziomu jak *urllib*, *telnetlib*, *imaplib* czy *ftplib* zapewniają obsługę poszczególnych protokołów. Nie jest to nic nadzwyczajnego. Wymagających zapewne zainteresuje coś innego – framework sieciowy *Twisted*, wspierający TCP, UDP, SSL/TLS, *multicast* czy protokoły takie jak HTTP, NNTP, IMAP, SSH, IRC, FTP. Framework ten umożliwi pisanie asynchronicznych klientów i serwerów różnego typu.

Test Driven Development

Testy jednostkowe i ciągła kontrola rozwijanej aplikacji w przypadku dużych projektów jest nieodzowna. Również w *Pythonie* znajdziemy narzędzia do przeprowadzania takich testów. Dostępnych jest kilka rozwiązań jak: *unittest*, *pymock*, *nose*, *nosy* czy *py.test*. *Pylons*, swój system testów jednostkowych oparł o *nose*, a *Django* o *doctest* i *unittest*, dodając dodatkowo własny system do testowania elementów kodu powiązane z bazą danych (*fixtures*). Można również stosować aplikacje takie jak *Selenium*.

Podsumowanie

Powyższy artykuł miał za zadanie przybliżyć programistom PHP alternatywne rozwiązania, jakie można wykorzystać przy tworzeniu stron internetowych. Zarówno PHP, jak i *Python*, mają swoje wady i zalety, lecz oba języki dobrze sprawdzają się w powierzonych im zadaniach. Jeżeli chcesz poznać nowy język o szerokich możliwościach, nie odchodząc zarazem od tworzenia stron www to, moim zdaniem, *Python* będzie dobrym wyborem.

PIOTR MALIŃSKI

Autor jest studentem Politechniki Warszawskiej na kierunku Technologia Chemiczna. Od 4 lat zajmuje się tworzeniem aplikacji w PHP, jak też od niedawna aplikacji o Pythonie.

Kontakt z autorem: riklaunim@gmail.com