${Pylons} 0.9.7 - Przewodnik część 1

Jaki celu przyświeca temu kursowi ?

Kiedy zaczynałem swoją przygodę z *Pylons* musiałem po całej sieci szukać mnóstwa użytecznych informacji. Ciężko było mi znaleźć stronę czy kurs, w którym pokazana byłaby większość z nich. Postanowiłem więc zebrać wszystko to co przydało się w pierwszym zderzeniu z tym frameworkiem, z różnych zakątków sieci i umieścić w tym krótkim przewodniku. Nie jest to kompendium, raczej przejście się po jego możliwościach i pokazanie tego co przydatne.

Gdzie szukać informacji ?

Ten krótki poradnik chcę zacząć od temat, w którym pokażę „gdzie szukać pomocy". Na pewno cennym źródłem jest dział *Biblioteki Pythona* poświęcony omawianemu przez nas frameworkowi[1]. Ze źródeł angielskojęzycznych nie można pominąć *Dokumentacji*[2] oraz *Pylons Book*[3]. Zauważ, że omawiany framework to zbitka różnych innych projektów, będących autonomicznymi rozwiązaniami tak więc w znacznej części szukając dokumentacji odwołujemy się do stron z opisami innych produktów takich jak na przykład *Mako*[4] czy *SQLAlchemy*[5]. Kiedy już to wiesz czas na start. Zaczynamy !

Instalacja.

> Jeżeli dystrybucja, której używasz posiada repozytorium z Pylons w deweloperskiej wersji skorzystaj z nich w pierwszej kolejności.

Opiszę instalację Pylons z użyciem narzędzia *easy_install*. Jest ona w miarę uniwersalna, oraz nie zależy od rodzaju dystrybucji. Dodatkowo pozwala nam pobrać najnowszą wersję frameworka. Pierwszym pytaniem jest: skąd wziąć *easy_install*? Powinniśmy poszukać w systemie paczki o nazwie zbliżonej do *setup_tools*. Po jej zainstalowaniu prawdopodobnie będziemy dysponować już wspomnianą wcześniej komendą. Aby pobrać Pylons w wersji 0.9.7 wydajemy w konsoli polecenie:

```
sudo easy_install Pylons==dev
sudo easy_install SQLAchemy>=0.4
sudo easy_install AuthKit
sudo easy_install Babel
```

Program pobierze kolejno wszystkie niezbędne pliki a następnie umieści je w naszym systemie. Gdy już skończy :) możemy zaczynać !

1 http://www.python.rk.edu.pl/w/p/pylonsindex/
2 http://docs.pylonshq.com/
3 http://pylonsbook.com/
4 http://www.makotemplates.org/docs/
5 http://www.sqlalchemy.org/docs/

Easy_install musi zostać wykonane z uprawnieniami administratora. Użycie powyżej polecenia *sudo* powinno to zapewnić. Jeżeli jednak w Twoim systemie nie ma polecenia o tej nazwie spróbuj użyć komendy *su*, podać hasło super użytkownika i wykonać powyższe linijki z pominięciem członu *sudo*. Inna możliwość to zalogowanie się w systemie jako super użytkownik i wpisanie *easy_install Pylons==dev && easy_install SQLAchemy>=0.4. && easy_install AuthKit && easy_install Babel*

Tworzymy pusty projekt.

Aby stworzyć pusty projekt przechodzimy do folderu, w którym chcemy umieścić nasze dzieło. U mnie będzie to katalog *Pylons* w folderze domowym użytkownika. Następnie z użyciem narzędzia *paster* tworzymy szkielet aplikacji. Moja będzie nazywała się *darc*.

```
cd Pylons
paster create -t pylons darc
```

Skrypt zada nam kilka pytań. Pierwsze o wykorzystywany mechanizm szablonów. Domyślnie proponowane jest *Mako*. Zgadzamy się wciskając przycisk *[Enter]*. Dalej skrypt chce wiedzieć czy stworzyć szkielet konfiguracji dla *SQLAlchemy*. Na początek może to wiele ułatwić. Wpisujemy *True* i klikamy *[Enter]*. Ponieważ nie chcemy aby nasz projekt posiadał właściwości charakterystyczne dla *Google App Engine* za trzecim razem odpowiadamy przecząco. Na końcu zostanie stworzony system folderów i skrypt poumieszcza w nich pliki, stanowiące szkielet pustej aplikacji.

Oczywiście możesz inaczej odpowiedzieć na te pytania w zależności od Twoich umiejętności, preferencji oraz przeznaczenia projektu.

Uruchamiamy nasz serwis.

Jeżeli chcemy zobaczyć nasze dzieło w przeglądarce musimy uruchomić serwer. Aby to zrobić przejdź do katalogu głównego naszej aplikacji a następnie, z użyciem narzędzia *paster*, wydaj polecenie:

```
cd ~/Pylons/darc
paster serve --reload development.ini
```

Aplikacja nie zakończy swojego działania. Serwer jest uruchomiony. Aby obejrzeć nasz projekt wpisujemy w przeglądarce internetowej *http://127.0.0.1:5000/*. Przywita nas domyślna strona startowa z bazowymi informacjami na temat *Pylons*. To na razie wystarczy. Jeżeli chcesz go zamknąć - wciśnij kombinację klawiszy *[CTRL]+[C]*.

Nie przejmuj się ostrzeżeniami typu *deprecated*. Wynikają z faktu używania rozwojowej wersji frameworka.

Opcja *--reload* spowoduje, że serwer będzie automatycznie wykrywał zmiany w plikach i restartował się gdy je zarejestruje. Dzięki temu nie będziemy musieli samodzielnie uruchamiać go ponownie w celu obejrzenia w przeglądarce wprowadzonych przez nas zmian. Pamiętaj, że za każdym razem uruchamiając serwer musisz podać ścieżkę do pliku konfiguracyjnego u nas: *development.ini*

Tworzymy własną stronę główną.

Domyślnie wyświetlanym plikiem, który widzimy po połączeniu się z serwerem w przeglądarce jest *index.html*. Pliki takie jak statyczne strony HTML, skrypt JavaScript, szablony stylów czy grafikę umieszczane są w katalogu *public* naszej aplikacji. Tam również znajduje się nasz "winowajca".

Aby stworzyć swoją własną stronę główną wystarczy oczywiście stworzyć samodzielnie plik *index.html* i umieścić go w folderze *public*, jednak my skorzystamy z możliwości jakie oferują widoki. W tym celu zaczniemy od kontrolera.

Omówienie MVC[6].

Co to w ogóle jest *kontroler*. Otóż w modelu *MVC* (***M**odel*, ***V**iew*, ***C**ontroller*) jest to część odpowiedzialna za logikę. Skąd całe zamieszanie ? Przecież pisząc zwykły program w Pythonie nie używałem żadnych *kontrolerów*. Skoro już przy kodowaniu w Pythonie jesteśmy: stosowanie architektury *MVC* to właśnie tworzenie takiego programu Pythona podzielonego na części. W przypadku witryn internetowych wyników nie oglądamy na konsoli, ale w przeglądarce i to z użyciem mnóstwa technologii z Pythonem nie mających nic wspólnego: *XHTML*, *JavaScript*, *CSS*, *XML*, *AJAX*. Ktoś więc pomyślał "oddzielmy kod programu Pythona" (*kontroler* - ***c**ontroller*) od kodu strony internetowej i jej prezentacji: *XHTML*, *JS*, *CSS*, *XML* (w*idok* **-** ***view*)**. Tak więc *kontroler* to miejsce, w którym piszesz "program", a *widoki* to szablony, w które wkładasz dane. Jest jeszcze oczywiście wspomniany wcześniej *model* - będący na swój sposób opcjonalnym elementem tej architektury. Pojawia się, mówiąc w dużym uproszczeniu, w momencie, w którym mamy do czynienia z bazą danych lub jakimś innym źródło informacji. Tworzone są klasy Pythona, które mają reprezentować pewne elementy będące przedmiotem naszego programu. Zazwyczaj są to obiekty będące odzwierciedleniami w programie konkretne tabele w bazie danych. Cały ten podział pozwala utrzymać porządek w projekcie, w którym używamy więcej niż tylko jednej technologii, ale całego ich zestawu. Sprawia, że nasz kod nie wygląda jak spaghetti, gdzie miesza się ze sobą *JavaScript*, *Python*, *XHTML*, *CSS* i *SQL* ale jest podzielony na jasno określone elementy odpowiedzialne za swoje zadania. Czas stworzyć swój pierwszy kontroler.

Ciekawskim polecam przeczytanie "*Spojrzenie na MVC*"[7].

Warstwa logiki.

Zanim stworzymy nasz pierwszy kontroler usuńmy z pola widzenia *Pylons* plik index.html

6 http://pl.wikipedia.org/wiki/MVC
7 http://matrix.zhr.pl/blog/2007/10/25/spojrzenie-na-mvc/

```
cd ~/Pylons/darc/darc/public
mv index.html index.html.org
```

Teraz wygenerujmy kod *kontrolera*.

```
paster controller users
```

Polecenie to stworzy dwa pliki. Pierwszy to rzeczona klasa kontrolera drugi zaś służący do jego testowania. Kolejne części kursu będą skupiały się głównie na tym umieszczonym w folderze *controllers*.

Uruchom serwer i wpisz w przeglądarce *http://127.0.0.1:5000/users/index*. Twoim oczom pojawi się napis *Hello World*. Ale o co chodzi ?

Czas na troszkę teorii samych *kontrolerów*. Wyedytuj plik *users.py*.

Kontroler to z punktu widzenia składni Pythona - klasa. Odpowiedzialny jest za logikę witryny. Tak na przykład *users* będzie odpowiedzialny za obsługę użytkowników: rejestrację, logowanie, wylogowywanie. Gdyby zależało nam na zamieszczeniu w systemie mechanizmu newsów stworzylibyśmy na przykład *kontroler* o nazwie *news*. Nazwa musi być adekwatna do przeznaczenia. Każdy kontroler posiada *akcje*. Z punktu widzenia języka programowania są to metody klasy. Każda z nich powinna odzwierciedlać jakieś jedno działanie związane z czynnością wykonywaną w systemie. Tak na przykład dla wspomnianego już kontrolera *news* moglibyśmy napisać akcje: *create, delete*, *list*, *update*, *edit*, *comment*, *show.* Znalazłyby się w nich kod odpowiedzialny kolejno za: tworzenie wiadomości, usuwanie wiadomość, wylistowanie wszystkich wiadomości, uaktualnienie wiadomości, wyedytowanie wiadomości, skomentowanie jej czy wyświetlenie. A co się stało u nas ?

Pylons wygenerował kod *kontrolera* o nazwie *users* z domyślną metodą *index* wyświetlającą napis *Hello World*. Jak widać adres strony internetowej to nic innego jak *http://adres.naszej.strony/nazwa_kontrolera/nazwa_akcji*. Czas skonfigurować stronę główną. Aby to zrobić musimy zajrzeć do routera.

Konfiguracja strony głównej w routerze.

Router to mechanizm, który przekłada nasze łamane adresy na odwołanie się do odpowiednich klas i ich metod. Konfigurujemy te zachowania edytując plik *routing*.py znajdujący się w katalogu *config*. Dodajmy regułkę mówiącą o tym aby podczas wywołania adresu **/** wywołać metodę *index* kontrolera *users*.

```
# CUSTOM ROUTES HERE
map.connect('/', controller='users', action='index')
map.connect('/{controller}/{action}')
map.connect('/{controller}/{action}/{id}')
```

Zmienione lub dopisane przez nas linijki będę zaznaczał żółtym tłem. Od teraz po wpisaniu *http://127.0.0.1:5000/* zobaczymy to samo co po *wpisaniu http://127.0.0.1:5000/users/index.* Otóż napisaliśmy naszemu routerowi, że jeżeli zażądamy adresu **/** to chcemy aby wywołał akcję *index* z kontrolera *users*. No dobrze - wyświetlany jest *Hello World*. Użycie tutaj metody return nie jest

rozwiązaniem stosowanym na co dzień. Do dyspozycji mamy potężny mechanizm szablonów *Mako*. Czas go wykorzystać.

Nasz pierwszy widok.

Już wiesz, że używając funkcji *return* w kontrolerze można wyświetlić krótki napis. Jednak realnie do wyświetlania całych stron internetowych służą widoki. Stworzymy więc jeden. Wszystkie szablony (będę używał tej nazwy wymiennie ze słowem widok) trzymane są w katalogu *templates*. Dobrą praktyką jest tworzyć podkatalogi, dla każdego z kontrolerów osobna i związane z nim widok we wspólnym folderze. Na starcie stwórzmy więc katalog *templates/users.* Ponieważ tworzymy widok dla akcji *index* kontrolera *users* plik nazwiemy *index.xhtml*. Logiczne. Wyedytuj plik *templates/users/index.mko* i wpisz:

```
<h1>Indeks użytkowników</h1>
```

Mamy przygotowaną bardzo prostą stronę w HTML. Teraz trzeba jeszcze powiedzieć kontrolerowi aby zamiast zwracać *Hello World* wyrenderował (wyświetlił) nasz widok. W tym celu edytujemy plik *controllers/users.py* i zmieniamy metodę *index* tak, aby przypominała tą poniżej.

```
def index(self):
    # Return a rendered template
    #   return render('/template.mako')
    # or, Return a response
    return render('users/index.xhtml')
```

Co się stało ? Otóż kontroler zwrócił żądanie rendorowania pliku widoku. Brzmi magicznie. Kontroler powiedział, że chce wyświetlić plik widoku. Proste ? To idziemy dalej.

Jak widać w komentarzu Pythona proponowanym rozszerzeniem pliku widoku jest *.mako*. Mamy jednak zupełną dowolność w ich używaniu. Osobiście użyłem rozszerzenia, przy którym mój edytor nadal traktuje dokument jak dokument HTML i koloruje jego składnię. Przy użyciu *.mako* straciłbym tą wygodę i komfort oraz znaczną część wsparcia edytora dla plików witryn internetowych. Oczywiście nie stanowi problemu aby w bardziej zaawansowanych programach skonfigurować je do pracy z niestandardowymi rozszerzeniami.

Przekazujemy dane z kontrolera.

Kontroler po wykonaniu swoich "obliczeń", swojego "programu", może przekazać ich wyniki do widoku. Spróbujmy więc coś takiego zrobić. W tym celu ponownie wyedytujmy plik *controlers/users.py*.

```
def index(self):
    # Return a rendered template
    #   return render('/template.mako')
    # or, Return a response
    c.name = "d'Arc"
    return render('users/index.xhtml')
```

Co się stało ? Do obiektu *c* dodaliśmy nowe pole o nazwie *name*. Obiekt *c* jest obiektem globalnym, czyli zmienną, która jest dostępna zarówno w kontrolerze jak i w widoku. W ten sposób możemy przekazywać informacje. Czas na wyedytowanie szablonu. Wyświetlimy na liście naszą zmienną.

```
<h1>Indeks użytkowników</h1>
<ul>
    <li>${c.name}</li>
</ul>
```

Uruchom serwer i zobacz efekt pracy. Co się stało ? Kontroler zapisał do zmiennej globalnej *c* pole *name* i wywołał widok, zaś w nim wyświetliliśmy kawałek kodu HTML, w tym listę, z naszym nazwiskiem. Składnia *${c.name}* to żądanie wyświetlenia zawartości zmiennej, tej samej, której wartość przypisaliśmy w kontrolerze. Ta dziwna składnia z dolarem to system szablonów *Mako*.

Co więcej ?

Na dziś to już wszystko. Jeżeli jesteś nowy w MVC proponuję abyś dla oswojenia się z widokami, kontrolerami i innymi zwierzakami stworzył kilka nowych kontrolerów, w każdym po trzy akcje, które będą renderowały swoje szablony. Spróbuj przekazać im i wyświetlić jakieś dane. Bardzo ważne jest abyś przed następną częścią kursu korzystanie z widoków i kontrolerów, oraz ich edycja było dla Ciebie rzeczą naturalną. Te z pozoru banalne czynności są podstawą. Im lepiej je opanujesz tym szybciej nauczysz się Pylons. Powodzenia !

Licencja

Spis treści