Wykład 4: Podstawy protokołu `HTTP` i Django

Date: 2015-11-01
tags: zaj4, wykład, materiały
category: materiały

> **Note**
> Wykład do pobrania również w wersji PDF.

Spis treści

Protokół `HTTP`

`HTTP` jest protokołem, za pomocą którego przeglądarka (albo inny klient) łączy się z serwerem w celu pobrania dokumentu.

> **Note**
>
> Będę mówił o protokole `HTTP` w wersji `1.1`, jest nowy protokół `HTTP/2`, który rozwiązuje wiele probemów, które `HTTP/1.1` miał.
> Zasadniczo `HTTP/2` nie zmienia semantyki `HTTP/1.1`, zmienia tylko formaty wysyłanych danych (np. jest binarny), tym co jest dla Was ważne jest **semantyka** dokładne szczegóły implementacji protokołu są przed programistami ukryte.

By zadać pobrać dane z serwera należy:

1. Otworzyć połączenie `TCP` do serwera na port, na którym nasłuchuje serwer (domyślnie `80`).
2. Wysłać treść zapytania.
3. Odczytać odpowiedź.

Przykładowa treść zapytania (uwaga na dole są dwie puste linie):

```
GET /webaplikacje/ HTTP/1.1
Host: db.fizyka.pw.edu.pl
```

> **Note**
>
> W drugiej linicje podaje nagłowek `Host` informujący serwer HTTP, że łączę się z hostem o nazwie: `db.fizyka.pw.edu.pl`. Muszę podać ów nagłowek, ponieważ na poziomie warstwy TCP łączę się już nie z domeną (nie z `db.fizyka.pw.edu.pl`), a z konkretnym adresem IP (`194.29.174.28`), jeden adres IP może obsługiwać wiele domen, więc w takim wypadku należy serwerowi powiedzieć: "łączę się z taką domeną).

Zapytanie to pobiera główną stronę przedmiotu webaplikacje.

Odpowiedź serwera wygląda tak:

```
HTTP/1.1 200 OK
Date: Sun, 01 Nov 2015 15:28:46 GMT
Server: Apache/2.2.15 (Red Hat)
Last-Modified: Thu, 29 Oct 2015 17:31:03 GMT
ETag: "1824931b-3a98-52341acd7169d"
Accept-Ranges: bytes
Content-Length: 15000
Connection: close
Content-Type: text/html; charset=UTF-8

<!DOCTYPE html>
<<Treść strony>>
```

Zapytanie HTTP

Ogólnie zapytanie ma następującą strukturę:

```
<<METHOD>> <<PATH>> HTTP/1.1
<<Lista nagłówków, każdy w nowej linii>>
<<pusta linia oddziela nagłówki od treści>>
<<Opcjonalnie treść>>
```

- `METHOD` to tzw. `metoda`, która określa **co serwer ma zrobić**,
- `PATH` definiuje **zasób**, na którym zostanie wykonana operacja
- Nagłówki pozwalają np. na negocjacje języka strony
- Treść zapytania pozwala przesłać dane do serwera.

Działanie metod

> **Note**
> Serwer może wykonać inne operacje od tych opisanch tutaj po odtrzymaniu zapytania, które wykorzystuje daną metodę. Czasem jest to przydatne, czasem prowadzi do problemów.

`GET`

: Pobiera zasób związany z daną ścieżką z serwera. Nie zawiera treści.

`OPTIONS`

: Działa jak `GET`, ale serwer nie zwraca treści (zwraca same nagłowki), może być to ważne, kiedy szukamy informacji w nagłowkach, ale nie chcemy pobierać całego dokumentu.

`PUT`

: Prosi serwer, by umieścić dane przesłane w treści zapytania pod daną ścieżką.

`DELETE`

: Prosi serwer by usunąć zasób znajdujący się pod daną ścieżką.

`POST`

: Prosi serwer by program powiązany z daną ścieżką przetworzył treść zapytania.

Note

Różnica między `PUT` a `POST` może być nieoczywista. W skrócie:

1. Kiedy chcę wysłać obrazek na serwer, tak by był on widoczny na ścieżce `/foo`, wysyłam zapytanie `PUT` (oczywiście serwer powinien sprawdzić czy mam prawa dostępu).
2. Kiedy chcę się zalogować wysyłam zapytania `POST` prosząc, by program skojarzony ze ścieżką `/login` przetworzył moje zapytanie.

Zapytania bez skutków ubocznych

Zapytania `GET` i `OPTIONS` nie powinny mieć skutków ubocznych, tj. ich przetworzenie nie powinno zmieniać stanu serwera.

Zapytania `PUT` oraz `DELETE` powinny być idempotentne, tj. stan serwera po przetworzeniu jednego zapytania powinien być taki sam, jak po przetworzeniu wielu tych samych zapytań (np. po wysłaniu zapytania `DELETE` zasób jest kasowany, więc nie ma go na serwerze, ponowienie tego zapytania nie powinno już w żaden sposób zmienić stanu serwera). `GET` oraz `OPTIONS` również powinny być idempotentne.

Note

Powyższe dwa warunki (brak skutków ubocznych i idempotentność) są dość ważne, w szczególności ważne jest, by: **Wykonanie zapytania GET nie powinno mieć skutków ubocznych**. Różne części infrasktuktury mogą mieć zakładać, że wykonanie zapytania `GET` nie zmienia stanu serwera, np:

- Wynik zapytania `GET` może zostać zapamiętabny przez przeglądarkę, lub serwer pośredniczący.
- Przeglądarka może w tle pobrać linki ze strony, tak by wyświetlić je użytkownikowi szybciej (teraz w praktyce nikt tak nie robi, ale w przeszłości były rozwiązania implementujące taką funkcjonalność).

Nagłówki

Nagłówki mają sporo zastosowań, ogólnie określają metadane zapytania, np:

1. Służą do negocjacji języka.
2. Służą do negocjacji formatu danych który wyśle serwer (serwer czasem może wysać te same dane w kilku formatach).
3. Pobierania fragmentu żądanego zasobu.

Lista jest długa

Odpowiedzi HTTP

Po przetworzeniu zapytania serwer wysyła odpowiedź, odpowiedź zawiera:

- Status
- Listę nagłówków
- Treść

Status

Status określa sposób realizacji zapytania klienta. Statusy są trzycyfrowymi liczbami, z których pierwsza cyfra określa **klasę**, odpowiedzi.

Lista klas jest następująca:

`1XX`

: Typowo rzadko używane.

`2XX`

: Sukces.

`3XX`

: Serwer informuje użytkownika, że musi wykonać dodatkową czynność, by zakończyć przetwarzanie. Np. przejść na kolejną stronę (która jest wskazana w nagłowkach odpowiedzi)

`4XX`

: Błąd po stronie klienta --- zapytanie źle sformuowane.

`5XX`

: Błąd po stronie serwera.

Przykładowe konwersacje `HTTP`

Pobranie zasobu

Skuteczne pobranie zasobu:

- Użytkownik wysyła zapytanie `GET` z prośbą pobranie zasobu po adresem `/foo`
- Serwer odpowiada ze statusem `200` i wysyła zawartość

Brak zasobu:

- Użytkownik wysyła zapytanie `GET` z prośbą pobranie zasobu po adresem `/foo`
- Serwer odpowiada ze statusem `404` informującym o braku wskazanego zasobu.

Przekierowanie

- Użytkownik wysyła zapytanie `GET` z prośbą pobranie zasobu po adresem `/foo`
- Serwer odpowiada ze statusem `307` informującym o braku wskazanego zasobu, oraz informującą że zasób jest pod adresem `/bar`

- Użytkownik wysyła zapytanie `GET` z prośbą pobranie zasobu po adresem `/bar`
- Serwer odpowiada ze statusem `200` i wysyła zawartość

Zalogowanie się do serwera:

- Użytkownik wysyła zapytanie `GET` z prośbą pobranie zasobu po adresem `/login`
- Serwer odpowiada formularzem logowania
- Użytkownik wyspełnia formularz i wysyła login i hasło jako zapytanie `POST`
- Serwer odpowiada statusem `200`, w odpowiedzi wysyłając formularz logowania oraz informację o błędym haśle.
- Użytkownik wysyła poprawne dane
- Serwer odpowiada statusem `307` z przekierowaniem na adres `/bar`
- Użytkownik pobiera stronę `/bar`.

> **Note**
> Jak działa mechanizm logowania wyjaśnię na kolejnych zajęciach.

Serwowanie dynamicznej zawartości

Serwery HTML mogą serwować całkowicie statyczną zawartość, np. strona przedmiotu składa się z plików HTML (oraz obrazków, plików css itp), a serwer tylko wysyła te pliki do klienta.

Czasem jest to rozwiązanie dobre, czasem nie.

> **Note**
> Głowne zalety takich stron to:
> - trudno jest się na taką stronę włamać
> - są bardzo wydajne (serwer pobiera plik i wysyła go do klienta)
> - nie wymagają utrzymania (jeśli mam bloga na WordPressie to muszę go aktualizować, jeśli tego nie zrobię ktoś może się włamać).

Serwery, które robią więcej niż serwowanie plików nazywamy serwerami **dynamicznymi**.

Common Gateway Infrastructure

CGI to najbardziej prymitywny standard serwowania zawarto±ci dynamicznej. Jest on wspierany zasadniczo wszędzie.

CGI działa mniej więcej tak:

1. Serwer otrzymuje zapytanie zdefiniowane jako wykonywane przez program CGI.
2. Serwer wykonuje pewien program umieszczaj¡c wcze±niej parametry zapytania w jako zmienne ±rodowiskowe.

3. Program na swoje standardowe wyjście zwraca odpowied1 dla użytkownika.

Zalety CGI

Działa zawsze i wszędzie, tj:

- (prawie) Każdy serwer WWW ma wsparcie dla CGI
- W każdym języku da się oprogramować CGI.

Wady CGI

- Powoduje uruchomienie **nowego procesu** dla każdego zapytania
- Pewne detale CGI zależą od sysemu operacyjnego.

> **Note**
> Przeciętna aplikacja w Django ładuje się 1sek (tj. jest to czas między uruchomieniem aplikacji, a przetworzeniem pierwszego zapytania), CGI spowodowałoby Django powoduje że czas obsłużenia danego wydłuża się o jedną sekundę (do każdego zapytania uruchamiamy Django "od zera").
> To opóźnienie jest nieakceptowalne, więc w systemach produkcyjnych bardzo rzadko korzysta się z CGI.

Jak nie CGI to co

Każdy język ma swoje rozwiązania, które zastępują CGI, każde z tych rozwiązań działa podobnie:

- Utrzymywana jest pula wątków (i procesów), z których każdy wątek jest w stanie wykonywać zapytania.
- Gdy serwer HTTP otrzyma zapytanie przekaże je do wykonania do jednego z wątków z puli. Jeśli wszystkie wątki będą zajęte, serwer poczeka, aż któryś się nie zwolni.

> **Note**
> Jak konfigurować Django do pracy z serwerem HTTP powiem na którychś kolejnych zajęciach.

HTML

HTML (Hyper Text Markup Language) jest językiem, w którym dostarczana jest większość zawartości w sieci WWW.

Zakładam, że Państwo znacie podstawy HTML.

Najprostsza strona HTML wygląda następująco:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/html">
<head>
    <meta charset="utf-8">
</head>
<body>
  <p>Hello world!
</body>
</html>
```

Ważne elementy:

`<!DOCTYPE html>`

: Deklaracja typu, mówi przeglądarce, że dokument jest w HTML w wersji 5.

`head`

: Sekcja head zawiera (niewyświetlane) metadane strony. Np. kodowanie.

`body`

: Zawiera zawartość strony.

W sekcji body mogą pojawiać się taki takie jak:

`<p>Treść</p>`

: Zawiera akapit tekstu

`<h1>Treść</h1>`, `<h2>Treść</h2>` ... `<hN>Treść</hN>`

: Zawiera nagłówek sekcji poziomu N.

> **Note**
> O tym jak robić **ładne** strony HTML powiemy później.

Formularze HTML

Do wysyłania danych do serwera służą formularze HTML, służą one do wysyłania danych do serwera.

```
<form action='/foo' method='POST'>
  <label>
    Username: <input name='username' type='text' placeholder='Username'/> <br/>
  </label>
  <label>
    Password: <input name='password' type='password' placeholder='Password'/>
  </label>
  <button>Submit</button>
</form>
```

`action`

Określa na jaką ścieżkę będzie wysłane zapytanie (domyślnie jest to **ta sama ścieżka**, z której pobrano dokument z formularzem).

`method`

Określa metodę zapytania.

Pojedyńcze pole formularza jest definiowane przez tag `input`, zawiera on następujące informacje:

`name`

Wartość wpisana w to pole zostanie wysłana jako parametr zapytania o wartości podanej jako wartość atrybutu name tagu input.

Przykładowo, jeśli w formularzu jest pole:

```
<input name='username' type='text' placeholder='Username'/>
```

w które użytkownik wpisał treść: "jbzdak" to do aplikacji jako parametr `username` zapytania zostanie wysłana wartość `jbzdak`.

`type`

Typ danych które dane pole przyjmuje.

> **Note**
> Dane z formularza zawsze docierają do aplikacji jako ciągi znaków, określenie typu w HTML ma na celu tylko ułatwienie użytkownikowi wpisania poprawnych danych.
>
> > **!DANGER!**
> > Uwaga: nawet jeśli w tagu `<input>` zdefiniowaliśmy, że wartość jest liczbą z zakresu od 1 do 10, **po stronie aplikacji i tak należy to sprawdzić ponownie**. Użytkownik może łatwo wyłączyć walidację formularzy HTML, a złośliwy użytkownik, może wykonać zapytanie z pominięciem HTML.
> > **Nigdy nie ufajcie danych pochodzącym od użytkownika**.

Tagi `label` służą do dodania opisów do pól formularza.

Linki

Do stworzenia łącza do innego zasobu służy tag `<a href="adres">Opis</a>`, tag ten spowoduje wysłanie przez przeglądarkę zapytania `GET`.

Django

Model View Template

Django posługuje się wzorcem Model-View-Template, gdzie:

`Model`

Jest to warstwa odpowiedzialna za odbieranie danych z bazy danych.

`View`

Jest to warstwa interpretująca zapytania HTTP.

`Template`

Jest to warstwa, która służy do generowania plików HTML.

W tym tygodniu zajmiemy się tylko Widokami.

Instalacja

By zaistalować `Django` należy:

- Pobrać Pythona 3.4
- Stworzyć środowisko wirtualne
- Aktywować je
- Napisać `pip install Django==1.8.0` (pracujemy na tej wersji Frameworku).

Rozpoczęcie pracy

Razem z instalacją Django instaluje się polecenie `django-admin`, by w aktyalnym katalogu stworzyć projekt należy napisać `django-admin startproject <<nazwa>>`, np: `django-admin startproject zaj3`.

Wewnątrz katalogu zaj3 znajduje się szkielet projektu, zawiera on między innymi skrypt `./manage.py`, który służy do zarządzania projektem, zawiera np. serwer deweloperski Django. By go uruchomić należy wpisać `./manage.py runserver`, a następnie otworzyć stronę `http://localhost:8000/`.

> **Note**
> Nie należy używać serwera deweloperskiego django na systemach produkcyjnych.

Pierwsza aplikacja

Aplikacja w Django jest zbiorem powiązanych: szablonów, modeli i widoków, które dostarczają pewną zamkniętą funkcjonalność. Przykładem aplikacji jest np. obsługa logowania.

By stworzyć aplikację należy wykonać polecenie: `./manage.py startapp zaj3app`.

Mając już gotowy projekt oraz aplikację, możemy napisać funkcję widoku.

Funkcje widoku w Django

Funkcje widoku w Django (z reguły są zdefiniowane wewnątrz pliku `views.py` w aplikacji) są zwykłymi funkcjami Pythona. Mają one ustalogą sygnaturę:

```
def hello_world(request):
  return HttpResponse(content='Hello World')
```

Trzeba jeszcze poinformować Django że: "Drogie Django chcę by na zapytanie na ścieżkę `/hello` odpowiadała funkcja `hello_world`". By to zrobić nalezy otworzyć plik `urls.py` z katalogu `zaj3` i wpisać dam odpowiednie odwzorowanie.

> **Note**
> Ogólnie: w każdym projekcie Django jest "root urlconf", głowny plik definuje to, które widoki są przypisane do jakich ścieżek, z reguły plik ten zawiera odwołania do plików `urls.py` z poszczególnych aplikacji, to który plik pełni funkcję "root urlconf" jest konfigurowalne.

Po modyfikacji plik `urls.py` powinien zawierać:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
  url('^hello/?$', views.hello_world)
]
```

Stała `urlpatterns` definiuje odwzorowanie ze ścieżki zapytania na funkcje, która owo zapytanie realizuje. Ścieżka jest podana za pomocą wyrażenia regularnego (działanie wyrażeń regularnych wyjaśnię później), na razie by Wasz widok odpowiadał na ścieżkę `/foo` w pliku `urls.py` musicie podać `^foo/^$`.

Przetwarzanie zapytań

Powiedzmy że chcemy by nasza strona:

1. Pytała użytkownika o imie
2. Wyświetlała "Cześć Imie".

By to zrobić trzeba wykonać dwa widoki:

```
ASK_TEMPLATE= """
  <!DOCTYPE html>
  <html xmlns="http://www.w3.org/1999/html">
  <head>
      <meta charset="utf-8">
  </head>
  <body>
    <h1>Podaj Imię<h1>
    <form action="greet" method="GET">
      <input name="name">
      <button> Submit</button>
```

```
    </form>
  </body>
  </html>
"""

def get_name(request):
  if request.method != 'GET': # Jeśli zapytanie nie używa GET
    return HttpResponse(status=405) # Zwracamy błąd użycia niedozwolonej metody
  return HttpResponse(content=ASK_TEMPLATE)

def greet_by_name(request):
  if request.method != 'GET':
      return HttpResponse(status=405)
  return HttpResponse(content="Witaj {}!".format(request.GET['name']))
```

Pierwszy widok (`get_name`) po prostu zwraca zawsze tego samego HTML zawierającego formularz z pytaniem o imie.

> **Note**
>
> Oczywiście w poważnych aplikacjach Django kod HTML nie jest wklejany jako stała do pliku z widokami, na kolejnych zajęciach pokażę, jak się pracuje z szablonami Django.

Drugi widok jest ciekawszy: pobiera on parametr zapytania przesłany z formularza. Obiekt `request` jest typu `django.http.request.HttpRequest` odwzorowuje on wszystkie własności zapytania HTTP, np. metoda zapytania jest dostępna za pomocą atrybutu: `request.method`.

Parametry zapytania `GET` dostępne są w słowniku `request.GET` a parametry zapytania `POST` dostępne są w słowniku `request.POST`