Rozwiązania szkieletowe w tworzeniu aplikacji www

Tomasz Łukaszuk

Katedra Oprogramowania
Wydział Informatyki
Politechnika Białostocka

Temat wykładu:

MVC - Kontroler, dyspozytor url

Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską
NCBR Narodowe Centrum Badań i Rozwoju

Materiały przygotowane w ramach projektu "PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji" (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

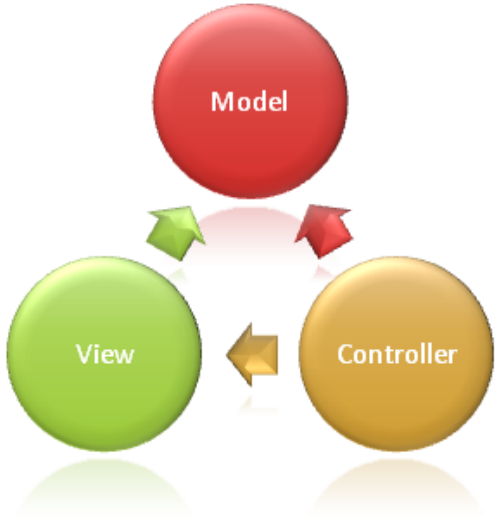
MVC - Kontroler, dyspozytor url: Agenda

1. MVC - Kontroler
2. Kontrolery i URLe w Django
   - Kontrolery
   - Dyspozytor URL
   - Moduły URLconf
   - URL reversing
3. Kontrolery i routing RoR
   - Kontrolery
   - Cel routingu
   - Podstawy routingu w RoR
   - Resource Routing
   - Inne metody routingu
   - Testowanie routingu

MVC - Kontroler

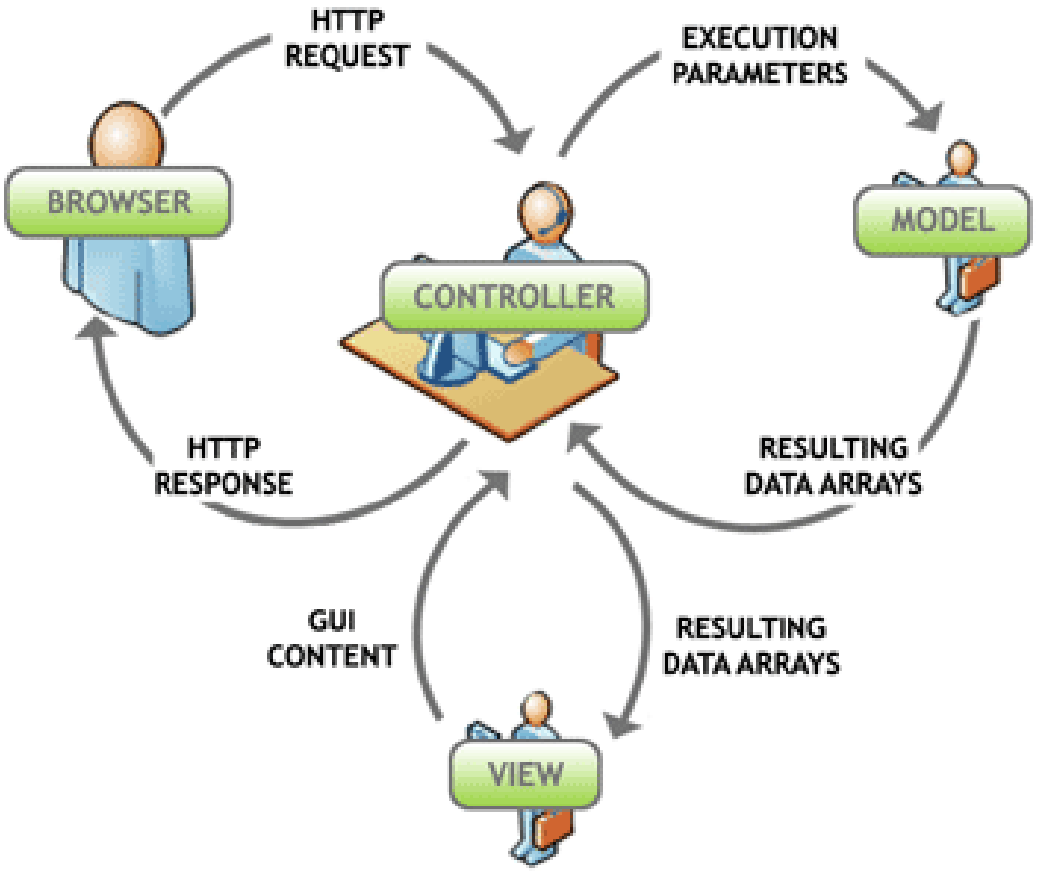
MVC: Kontroler - przypomnienie

- zadaniem kontrolera jest odbiór, przetworzenie oraz analiza danych wejściowych
- może zmienić stan modelu, odświeżyć widok, przełączyć sterowanie na inny kontroler



- może istnieć wiele kontrolerów, w danym momencie tylko jeden z nich steruje aplikacją
- kontroler posiada bezpośrednie wskazania na modele i widoki, z którymi współpracuje
- jest zasadniczą częścią każdej biblioteki implementującej MVC
- posiada akcje, tj. czynności wykonywane przez aplikację i określające jaki widok należy wyświetlić

Zadania kontrolera



1. odbiór żądania
2. decyzja o realizacji żądanych operacji określonych w parametrach żądania
3. rozdysponowanie zadań
4. określenie widoku do zwrócenia w odpowiedzi na żądanie
5. zwrot odpowiedzi

Kontrolery i URLe w Django

Kontrolery

Django - views.py

```
1  from django.http import HttpResponse
2  from django.shortcuts import render
3  import datetime
4
5  def current_datetime(request):
6      now = datetime.datetime.now()
7      html = "<html><body>It is now %s.</body></html>" % now
8      return HttpResponse(html)
9
10 def detail(request, poll_id):
11     try:
12         p = Poll.objects.get(pk=poll_id)
13     except Poll.DoesNotExist:
14         raise Http404
15     return render(request, 'polls/detail.html', {'poll': p})
```

Dyspozytor URL

Dyspozytor URL

- Czysty, elegancki schemat URL jest ważnym szczegółem w wysokiej jakości aplikacji internetowej
- Django pozwala projektować URL w dowolnej postaci bez ograniczeń narzucanych przez framework.
- W adresach URL nie jest wymagane używanie .php lub .cgi, ani takich nonsensów jak 0,2097,1-1-1928,00 (jak np. na www.wp.pl)
- *Cool URIs don't change* by World Wide Web creator Tim Berners-Lee http://www.w3.org/Provider/Style/URI

Jak Django przetwarza żądanie

- Określenie głównego modułu URLconf - wartość ustawienia ROOT_URLCONF lub atrybut `urlconf` określony w obiekcie HttpRequest reprezentującym żądanie (domyślnie: `urls.py` w katalogu administracyjnycm projektu django)
- Załadowanie głównego modułu URLconf i odnalezienie zmiennej `urlpatterns`
- Odnalezienie pierwszego wzorca URL pasującego do żądanego URLa. Metoda żądania (GET, POST etc.) i parametry są ignorowane (ale przekazywane do funkcji widoku)
- Import i wywołanie funkcji widoku przypisanej do pasującego wzorca URL

Moduły URLconf

Przykładowy URLconf

```
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path('articles/2003/', views.special_case_2003),
7      path('articles/<int:year>/', views.year_archive),
8      path('articles/<int:year>/<int:month>/', views.month_archive),
9      path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),
10 ]
```

Uwagi:

- Pierwszy import udostępnia funkcję `path`
- Aby przechwycić wartość z adresu URL, wystarczy umieścić nawiasy wokół elementu reprezentującego wartość
- Nie trzeba dodawać wiodącego ukośnika (`articles`, a nie `/articles`)
- Kolejność wzorców URL jest istotna!

Przykładowy URLconf

```
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      path('articles/2003/', views.special_case_2003),
7      path('articles/<int:year>/', views.year_archive),
8      path('articles/<int:year>/<int:month>/', views.month_archive),
9      path('articles/<int:year>/<int:month>/<slug:slug>/', views.article_detail),
10 ]
```

- Żądanie `/articles/2005/03/` pasuje do trzeciego wpisu z listy. Django wywoła funkcję `views.month_archive(request, year=2005, month=3)`.
- `/articles/2003/` pasuje do pierwszego wpisu. Zostanie wywołana funcja `views.special_case_2003(request)`.
- `/articles/2003` nie pasuje do żadnego wpisu, ponieważ każdy wpis wymaga obecnosci znaku `/` na końcu.

Użycie wyrażeń regularnych

```
1  from django.urls import path, re_path
2
3  from . import views
4
5  urlpatterns = [
6      path('articles/2003/', views.special_case_2003),
7      re_path(r'^articles/(?P<year>[0-9]{4})/$', views.year_archive),
8      re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/$',
9              views.month_archive),
10     re_path(r'^articles/(?P<year>[0-9]{4})/(?P<month>[0-9]{2})/(?P<slug>[\w-]+)/$',
11             views.article_detail),
12 ]
```

Funkcja `include()`

```
1  from django.urls import include, path
2
3  urlpatterns = [
4      # ... snip ...
5      path('community/', include('aggregator.urls')),
6      path('contact/', include('contact.urls')),
7      # ... snip ...
8  ]
```

- `include()` zasadniczo osadza zestaw innych URLi

Funkcja `include()`

```
1  from django.urls import include, path
2
3  from apps.main import views as main_views
4  from credit import views as credit_views
5
6  extra_patterns = [
7      path('reports/', credit_views.report),
8      path('reports/<int:id>/', credit_views.report),
9      path('charge/', credit_views.charge),
10 ]
11
12 urlpatterns = [
13     path('', main_views.homepage),
14     path('help/', include('apps.help.urls')),
15     path('credit/', include(extra_patterns)),
16 ]
```

- Użyj pojedynczych cudzysłowów aby dołączyć moduł URLconf
- Pomiń cudzysłowy aby dołączyć listę wzorców

`include()` i przechwytywane parametry

Dołączany URLconf otrzymuje wszystkie parametry z macierzystego URLconf:

```
 1   In settings/urls/main.py
 2  from django.urls import include, path
 3
 4  urlpatterns = [
 5      path('<username>/blog/', include('foo.urls.blog')),
 6  ]
 7
 8  # In foo/urls/blog.py
 9  from django.urls import path
10  from . import views
11
12  urlpatterns = [
13      path('', views.blog.index),
14      path('archive/', views.blog.archive),
15  ]
```

- Zarówno `blog.index` jaki i `blog.archive` otrzymają kluczowy parametr `username`

URL reversing

Podstawy URL reversing

Rozważmy następujący wzorzec URL:

```
1  from django.urls import path
2
3  from . import views
4
5  urlpatterns = [
6      #...
7      path('articles/<int:year>/', views.year_archive, name='news-year-archive'),
8      #...
9  ]
```

Jest wygodne, aby móc pobierać URL prowadzący do danej funkcji wioku. Można odwołać się do każdego wzorca indywidualnie przy użyciu jego nazwy.

Podstawy URL reversing (cd)

- W templatce: użyj `url` template tag:

```
1  {% url 'news-year-archive' 1945 %}
```

- W kodzie Pythona: użyj funkcji `reverse()`:

```
1  from django.http import HttpResponseRedirect
2  from django.urls import reverse
3
4  def myview(request):
5      return HttpResponseRedirect(reverse('news-year-archive', args=(2005,)))
```

Kontrolery i routing RoR

Kontrolery

Podstawy kontrolerów w RoR

- moduł Action Controller
- kontroler jest klasą języka Ruby dziedziczącą z `ApplicationController`
- kontrolery są zapisywane w plikach `app/controllers/name_controller.rb`
- routing określa, który kontroler powinien zostać użyty do wygenerowania odpowiedzi na żądanie klienta
- kontroler jest odpowiedzialny za przetworzenie żądania i wygenerowanie odpowiedniej odpowiedzi (wyjścia)

Metody i Akcje

- kontroler jest klasą języka Ruby dziedziczącą z `ApplicationController`
- kontroler posiada metody tak jak każda inna klasa
- Kiedy aplikacja odbiera żądanie, routing określa, który kontroler i akcja mają być uruchomione, Rails tworzy instancję kontrolera i uruchamia metodę o takiej samej nazwie jak nazwa akcji.

```
1  class ClientsController < ApplicationController
2    # akcje są metodami publicznymi
3    def new
4    end
5
6     # metody prywatne nie mogą być używane jako akcje
7     private
8
9     def foo
10    end
11 end
```

Cel routingu

Cel routingu - wprowadzenie

- HTTP (Hypertext Transfer Protocol)
  standard żądanie-odpowiedź w przetwarzaniu klient-serwer, klient wysyła żądanie, serwer wykonuje żądane operacje i zwraca odpowiedź
- metody żądań HTTP
  HEAD, GET, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATH
- standardowe adresy URLs
  http://host.com/resource.php?param=5
- URLs mapping
  łączenie adresu URL z kodem
  generowanie URL'i z kodu

Podwójny cel routing'u

- łączenie adresu URL z kodem

```
1  GET /patients/17
```

- generowanie URL'i z kodu

```
1  @patient = Patient.find(17)
2
3  <%= link_to "Patient Record", patient_path(@patient) %>
4
5  # http://example.com/patients/17
```

Podstawy routingu w RoR

Plik `routes.rb` I

- dwa komponenty odpowiedzialne za routing w Rails
  silnik routing'u i plik config/routes.rb
- format pliku routes.rb
  jeden duży blok przekazywany do AppName::Application.routes.draw,
  zawiera komentarze i linie kodu definiujące drogi routing'u w aplikacji

```
1  AppName::Application.routes.draw do
2     # The priority is based upon order of creation:
3     # first created -> highest priority.
4
5     # Sample of regular route:
6     #   match 'products/:id' => 'catalog#view'
7     ...
8     # Sample resource route (maps HTTP verbs to controller actions automatical
9     #   resources :products
10    ...
11 end
```

Plik `routes.rb` II

- typy zawartości routes.rb
  Drogi RESTful (Resource)
  Drogi nazwane
  Drogi regularne
- przetwarzanie routes.rb
  plik jest przetwarzany od góry do dołu
  żądanie zostanie wysłane do pierwszej pasującej drogi
  jeżeli nie ma pasującej drogi, Rails zwraca odpowiedź o statusie 404

Typu dróg routing'u w Rails

- Resource Routes

```
1  resources :products
2  resources :products do
3      resources :comments, :sales
4      resource :seller
5  end
```

- Drogi regularne

```
1  get 'products/:id' => 'catalog#view'
```

- Drogi nazwane

```
1  get 'exit', to: 'sessions#destroy', as: :logout
```

- Główna strona obsługiwana przez drogę "root"

```
1  root :to => "welcome#index"
```

Resource Routing

Wprowadzenie do RESTful

- obecnie obowiązujący standard routing'u w Rails
- założenia routing'u RESTful są zawarte w pracy doktorskiej Roy'a Fielding'a "Architectural Styles and the Design of Network-based Software Architectures"
- REST, akronim terminu Representational State Transfer
- dwie główne zasady REST
  - Korzystanie z identyfikatorów zasobów (URL) do reprezentowania zasobów
  - Przesyłanie reprezentacji stanów tych zasobów między elementami systemu

```
1  DELETE /photos/17
```

CRUD, typy żądań i akcje

żądanie HTTP - akcja kontrolera - operacja CRUD w bazie danych

```
1   resources :photos
```

żądanie	**URL**	**kontroler**	**akcja**	**cel użycia**
GET	/photos	Photos	index	wyświetlenie listy wszystkich zdjęć
GET	/photos/new	Photos	new	zwrócenie formularza do tworzenia zdjęcia
POST	/photos	Photos	create	tworzenie nowego zdjęcia
GET	/photos/1	Photos	show	wyświetlenie konkretnego zdjęcia
GET	/photos/1/edit	Photos	edit	zwrócenie formularza do edycji zdjęcia
PUT	/photos/1	Photos	update	aktualizacja zdjęcia
DELETE	/photos/1	Photos	destroy	usunięcie zdjęcia

Adresy URL i ścieżki dostępu

helper'y	akcje
`photos_url`, `photos_path`	index, create
`new_photo_url`, `new_photo_path`	new
`edit_photo_url`, `edit_photo_path`	edit
`photo_url`, `photo_path`	show, update, destroy

- `_url` helper generuje łańcuch znaków zawierający pełny adres URL
- `_path` helper generuje łańcuch zawierający ścieżkę dostępu odnoszącą się do katalogu głównego aplikacji

```
1  <%= link_to "List of photos", photos_url %>
2
3  photos_url      # => "http://www.example.com/photos"
4  photos_path     # => "/photos"
```

Dopasowanie zasobów I

- :controller (pozwala użyć kontrolera o innej nazwie niż dostępna od zewnątrz nazwa zasobu)

```
1  resources :photos, :controller => "images"
```

- :constraints (definuje ograniczenia, warunki)

```
1  resources :photos, :constraints => {:id => /[A-Z][A-Z][0-9]+/}
2  # photos/RR27
3
4  constraints(:id => /[A-Z][A-Z][0-9]+/) do
5      resources :photos
6      resources :accounts
7  end
```

- :as (zmienia nazwy helper'ów)

```
1  resources :photos, :as => "images"
2  # /images/1/edit
```

Dopasowanie zasobów II

- :path_names (zmienia nazwy wywołań new i edit)

```
1  map.resources :photos, :path_names => {:new=>'make', :edit=>'change'}
2  # /photos/make
3  # /photos/1/change
```

- :as (dodaje prefiks do nazw helperów)

```
1  scope "admin" do
2       resources :photos, :as => "admin_photos"
3  end
4  resources :photos
5  #helpery admin_photos_path, new_admin_photo_path etc.
```

- :only, :except (ogranicza tworzone drogi routing'u)

```
1  resources :photos, :only => [:index, :show]
2  resources :photos, :except => :destroy
```

Dopasowanie zasobów III

- :path (przetłumaczone nazwy dróg)

```
1  scope(:path_names => { :new => "neu", :edit => "bearbeiten" }) do
2       resources :categories, :path => "kategorien"
3  end
4  #/kategorien/neu
5  #/kategorien/:id/bearbeiten
```

Definiowanie kilku zasobów jednocześnie

```
1  resources :photos, :books, :videos
2
3  #Działa tak samo
4
5  resources :photos
6  resources :books
7  resources :videos
```

Pojedyncze zasoby

Pojedynczy zasób - zasób do którego klient zawsze może odwołać się bez ID

```
1   resource :geocoder
```

żądanie	**URL**	**akcja**	**cel użycia**
GET	/geocoder/new	new	zwrócenie formularza HTML do tworzenia geokodera
POST	/geocoder	create	tworzenie nowego geokodera
GET	/geocoder	show	wyświetlenie jednego jedynego geokodera
GET	/geocoder/edit	edit	zwrócenie formularza HTML do edycji geokodera
PUT	/geocoder	update	aktualizacja geokodera
DELETE	/geocoder	destroy	usunięcie geokodera

Routing a przestrzenie nazw kontrolerów

organizowanie grup kontrolerów w przestrzenie nazw, np. kontrolery administracyjne w przestrzeni `Admin::`, umieść te kontrolery w katalogu `app/controllers/admin`

```
1  namespace "admin" do
2     resources :posts, :comments
3  end
```

Żądanie	**ścieżka**	**akcja**	**helper**
GET	/admin/posts	index	admin_posts_path
GET	/admin/posts/new	new	new_admin_posts_path
POST	/admin/posts	create	admin_posts_path
GET	/admin/posts/1	show	admin_post_path(id)
GET	/admin/posts/1/edit	edit	edit_admin_post_path(id)
PUT	/admin/posts/1	update	admin_post_path(id)
DELETE	/admin/posts/1	destroy	admin_post_path(id)

Zasoby zagnieżdżone

zasoby, które są logicznie dziecmi innych zasobów

Listing 1: models

```
1  class Magazine < ActiveRecord::Base
2    has_many :ads
3  end
4
5  class Ad < ActiveRecord::Base
6    belongs_to :magazine
7  end
```

Listing 2: routes.rb

```
1  resources :magazines do
2    resources :ads
3  end
4  # /magazines/1/ads
5  # /magazines/1/ads/new
```

Generowanie drogi routingu z tablic

Oprócz helperów routingu, Railsy mogą też generować drogi RESTful z tablicy parametrów.

Listing 3: routes.rb

```
1  resources :magazines do
2    resources :ads
3  end
```

Listing 4: view file

```
1  <%= link_to "Ad details", magazine_ad_path(@magazine, @ad) %>
2  <%= link_to "Ad details", url_for(@magazine, @ad) %>
3  <%= link_to "Ad details", [@magazine, @ad] %>
4  <%= link_to "Magazine details", @magazine %>
```

Dodawanie większej liczby akcji RESTful

- 7 domyślnie tworzonych dróg routing'u.
- Jeśli chcesz możesz dodawać kolejne drogi członkowskie (member routes; odnoszące się do konkretnego zasobu), a także drogi tworzące zasoby oraz drogi odnoszące się do zestawu zasobów jako całości.

```
1  resources :photos do
2    member do
3      get 'preview'
4    end
5  end
6  # recognize /photos/1/preview
7  # create the preview_photo_url and preview_photo_path helpers
8
9  resources :photos do
10   collection do
11     get 'search'
12   end
13 end
14 # recognize /photos/search
15 # create the search_photos_url and search_photos_path route helpers
```

Inne metody routingu

Drogi regularne

- sposób na przetłumaczenie URL na odpowiednie kontrolery i akcje
- musisz samodzielnie ustawić każdą drogę
- możesz łączyć te dwa typy routingu (RESTful i regularny) wewnątrz jednej aplikacji

```
1  get ':controller/:action/:id'
2  # /photos/show/1
3  get ':controller/:action/:id/:user_id'
4  # /photos/show/1/2
5  get ':controller/:action/:id/with_user/:user_id'
6  # photos/show/1/with_user/2
```

- hash table `params`
- querystring

```
1  # /photos/show/1
2  params[:id] is set to 1
3  # /photos/show/1?user_id=2
4  params[:id] is set to 1 and params[:user_id] is set to 2
```

Drogi nazwane

Możesz określić nazwę dla drogi routing'u.

```
1  get 'exit' => 'sessions#destroy', :as => :logout
2  # create logout_path and logout_url
3  # calling logout_path will return /exit
```

Przekierowanie

Możesz przekierować dowolną ścieżkę do innej scieżki.

```
1  get "/stories" => redirect("/posts")
2  get "/stories/:name" => redirect("/posts/%{name}")
```

Pusta droga

- pusta droga - droga obsługująca żądanie zwrócenia głównej strony aplikacji
- root command

```
1   root :to => 'pages#main'
```

Testowanie routingu

Przeładanie istniejących dróg routingu poprzez Rake

```
1  rails routes
```

```
1             users GET  /users          {:controller=>"users",:action=>"index"}
2  formatted_users GET  /users.:format  {:controller=>"users",:action=>"index"}
3                  POST /users          {:controller=>"users",:action=>"create"}
4                  POST /users.:format  {:controller=>"users",:action=>"create"}
```