# Rozwiązania szkieletowe w tworzeniu aplikacji www

Tomasz Łukaszuk

Katedra Oprogramowania
Wydział Informatyki
Politechnika Białostocka

## Temat wykładu:

Konfiguracje i konwencje

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.

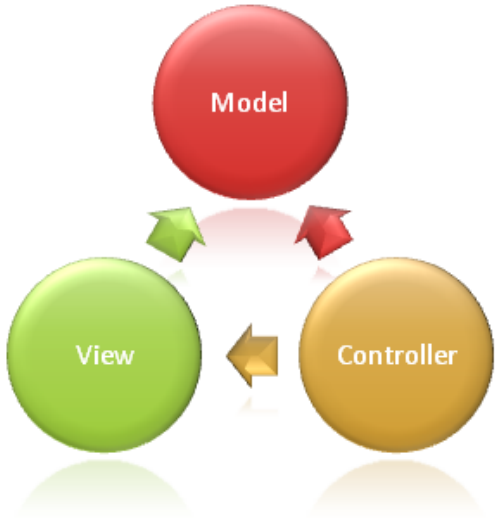
# Konfiguracje i konwencje: Agenda

1. Wzorzec projektowy MVC
2. Konfiguracje i konwencje w Django
   - Komponenty django
   - Instalacja
   - Projekt - tworzenie, budowa, ustawienia
   - Aplikacje django
   - Middleware
3. Konfiguracje i konwencje w RoR
   - Struktura frameworku
   - Struktura aplikacji
   - Ustawienia aplikacji
   - Rails Command Line Tools

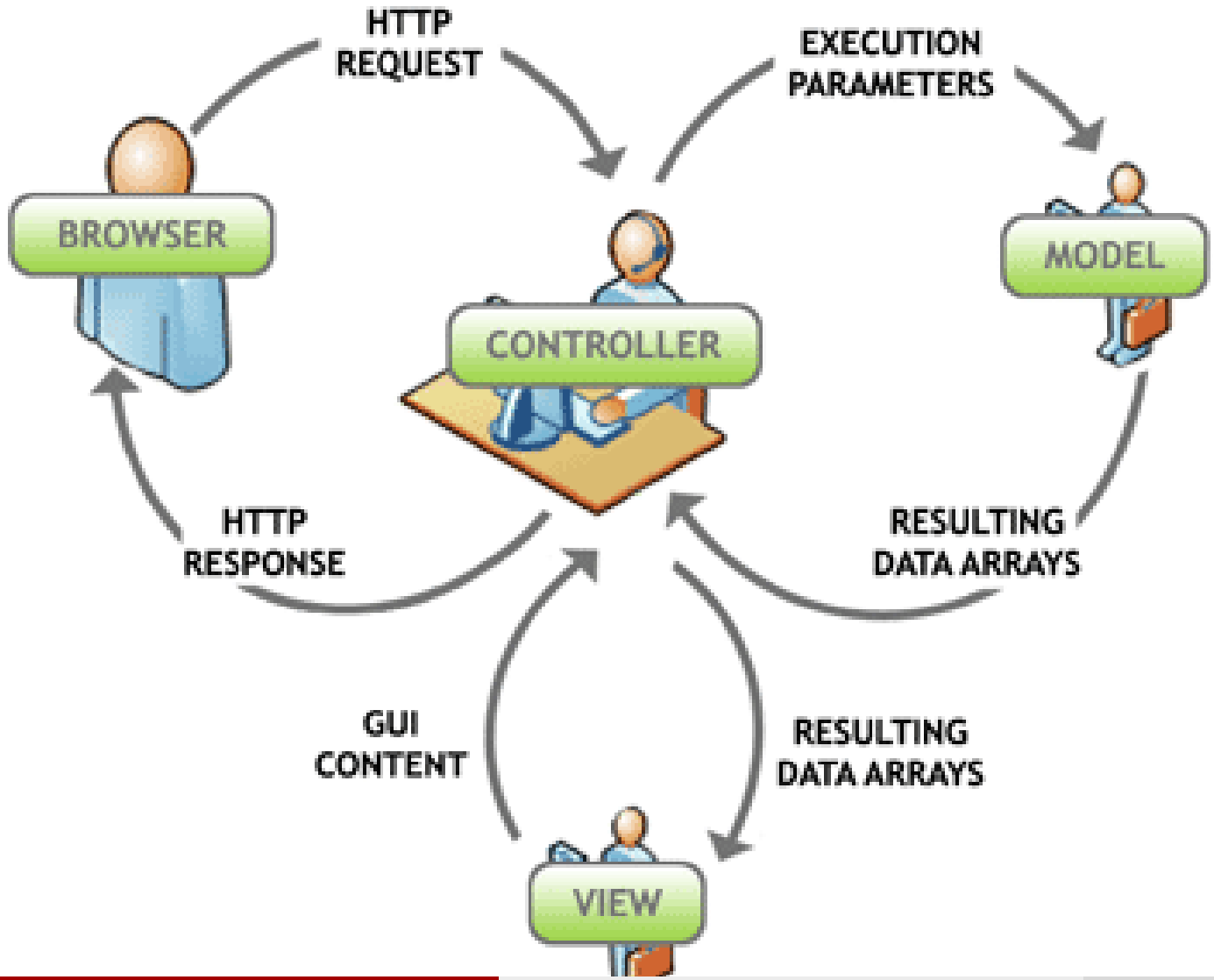
# Wprowadzenie

MVC = Model-View-Controller = Model-Widok-Kontroler

- architektoniczny wzorzec projektowy
- organizowanie struktury aplikacji posiadających graficzne interfejsy użytkownika
- może być także traktowany jako złożony wzorzec wykorzystujący idee wzorców prostych takich, jak Obserwator, Strategia, Kompozyt



# Obsługa żądania



# Django

# General


- Wysokopoziomowy pyhonowy framework webowy, który wspiera szybkie, czyste i pragmatyczne projektowanie
- Bazuje na architektonicznym wzorcu projektowym Model-View-Controller (MVC), ale “view” w django odpowiada kontrolerowi, zaś “template” w django odpowiada widokowi
- Pierwotnie opracowany do zarządzania kilkoma witrynami informacyjnymi World Company of Lawrence, Kansas
- Udostępniony publicznie w lipcu 2005 na licencji BSD
- Nazwany na cześć gitarzysty Django Reinhardt grającego muzykę gypsy jazz
- Od czerwca 2008 zarządzany przez Django Software Foundation
- Aktualna wersja stabilna: 5.1.7 (6 marca 2025)

# Komponenty django

- Lekki, samodzielny serwer WWW do używania na etapie rozwoju i testowania
- Serializacja i walidacja formularzy
- System cache'owania, który może używać dowolnego z kilku dostępnych metod obsługi pamięci podręcznej
- Wsparcie dla klas middleware, które mogą ingerować w przetwarzanie żądań
- Wewnętrzny system wysyłający, który pozwala komponentom aplikacji na informowanie o zdarzeniach za pomocą predefiniowanych sygnałów
- System internacjonalizacji
- System serializacji, który pozwala na generowanie i odczytywanie XMLowej i/lub JSONowej reprezentacji instancji modeli
- System do rozszerzenia możliwości silnika szablonów
- Interfejs do python'owych testów jednostkowych

# Luźne powiązania

Luźne powiązanie oznacza, że poszczególne komponenty Django są rozdzielone od siebie tak bardzo jak to możliwe. Na przykład:

- język szablonów Django jest bardzo przyjazny w reprezentowaniu zawartości modeli, ale może równie przyjemnie reprezentować inne obiekty języka python
- Jeśli nie podoba się komuś język szablonów Django, może go zastąpić np. Jinja2
- Django ORM jest łatwe w konfiguracji i dostępie do bazy danych, ale można użyć zamiast niego SQLAlchemy

Jeśli wolisz korzystać z narzędzi innych, niż domyślnie przewidziane w Django, możesz z nich korzystać.

# Instalacja

- zainstaluj Python
- zainstaluj serwer bazy danych (PostgreSQL, MySQL, Oracle lub SQLite) i odwpowiednie powiązania dla Pythona
- zainstaluj Django
  - ze źródeł: pobierz i rozpakuj Django-NNN.tar.gz, potem `setup.py install`
  - używając pip: `pip install Django==3.2`
- sprawdź obecność Django

```
1  >>> import django
2  >>> print(django.get_version())
3  3.2
```

# Tworzenie projektu

```
1   $ django-admin startproject mysite
```

- Projekt jest folderem (tutaj: `mysite`) zawierającym kolekcję ustawień instancji Django, m.in. konfigurację bazy danych, specyficznych opcji Django i specyficznych ustawień aplikacji
- Projekt może używać dowolną liczbę aplikacji

# Zawartość projektu

```
mysite/
    manage.py
    mysite/
        __init__.py
        settings.py
        urls.py
        wsgi.py
```

# Administracja: `django-admin.py` and `manage.py`

Użycie:

```
1  django-admin.py <subcommand> [options]
2  manage.py <subcommand> [options]
```

- `django-admin.py` to narzędzie command-line do wykonywania czynności administracyjnych.
- `manage.py` to wrapper do `django-admin.py`
  - dodaje bieżący projekt do `sys.path`.
  - ustawia zmienną środowiskową DJANGO_SETTINGS_MODULE tak aby wskazywała na `settings.py` w katalogu peojektu.
- *subcommand* wskazuje co zrobić. Przykłady:
  - `syncdb`: Tworzy tabele w bazie danych dla wszystkich aplikacji używanych w projekcie.
  - `shell`: Ustawia środowisko projektu i uruchamia interaktywny interpreter Pythona
- Aplikacje mogą rejestrować swoje własne akcje w `manage.py`
- Podczas pracy z pojedynczym projektem Django łatwiej jest używać `manage.py`

# Ustawienia projektu: `settings.py`

`settings.py` jest modułem Pythona, który zawiera wszystkie ustawienia konfiguracyjne dla całego projektu.
Użycie settings w kodzie Pythona:

```
1  from django.conf import settings
2
3  if settings.DEBUG:
4      # Do something
```

Ważne uwagi:

- Zawsze odwołuj się do settings przez import obiektu `django.conf.settings`
- Nie zmieniaj settings podczas pracy aplikacji/projektu
- ZAWSZE chroń plik settings.py przed nieautoryzowanym dostępem!

# Podstawowe ustawienia

- `DATABASES`: ENGINE, HOST, NAME, etc.: ustawienia dotyczące bazy danych
- `INSTALLED_APPS`: aplikacje Django, które są używane w bieżącym projekcie
- `MIDDLEWARE_CLASSES`: klasy middleware, które przetwarzają każde żądanie
- `ROOT_URLCONF`: główny plik `url.py`
- `TEMPLATE_DIRS`: bewzględna ścieżki do źródeł szablonów, w pożądku poszukiwania

Jest WIELE więcej ustawień, na szczęście Django dostarcza też ustawienia domyślne.

# Tworzenie aplikacji

```
1  $ django-admin startapp myapp
```

lub

```
1  $ python manage.py startapp myapp
```

- Aplikacje Django mogą istnieć w dolonym miejscu w systemie plików, ale ich lokalizacja musi być zapisana w `os.path`.
- Zawartość `myapp`:
  - `__init__.py`: Pusty plik wskazujący, że folder powinien być traktowany jak pakiet Pythona
  - `admin.py`: Konfigurowanie aplikacji admin.
  - `migrations/`: Miejsce składowania plków migracji.
  - `models.py`: Miejsce definiowania modeli aplikacji.
  - `tests.py`: Testy wywoływane na komendę `manage.py test`.
  - `views.py`: Funkcje widoków aplikacji.

# Funkcje widoków

- W Django widok jest funkcją/metodą, która potrzebuje przynajmniej jednego argumentu: `request`
- Widok powienien zwracać obiekt `HttpResponse`
- Widoki zwykle są definiowane w folderze aplikacji w pliku `views.py`

```
1  from django.http import HttpResponse
2
3  def hello_world_view(request):
4      return HttpResponse('Hello, world!')
```

- Zamiast podpinania widoku bezpośrednio do `urls.py` projektu, aplikacja powinna definiować własny `urls.py`

```
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.hello_world_view, name='hello_world'),
6  ]
```

# Mapowanie adresów url

URLe zdefionowane w `myapp/urls.py` moagą być dołączone do `urls.py` projektu w następujący sposób:

```
1  from django.urls import include, path
2  from django.contrib import admin
3
4  urlpatterns = [
5      path('myapp/', include('myapp.urls')),
6      path('admin/', admin.site.urls),
7  ]
```

Pozwala to na łatwe dołączanie aplikacji do różnych miejsc.
Zauważ różnicę między dwoma wywołaniami `include` (cudzysłów)!

# Aktywacja aplikacji

Aby aktywować aplikację w projekcie Django, należy dodać ją do ustawień `INSTALLED_APPS`:

```
1  INSTALLED_APPS = (
2      'django.contrib.auth',
3      'django.contrib.contenttypes',
4      'django.contrib.sessions',
5      'django.contrib.sites',
6      'myapp',
7  )
```

UWAGA: tutaj, aplikacja została stworzona wewnątrz katalogu projektu.

Aplikacje Django są "pluggable": Mogą być używane w wielu projektach, możliwe jest dystrybuowanie aplikacji, ponieważ nie muszą być związane z danym projektem Django.

# Aplikacje dostarczane z frameworkiem Django

- `django.contrib.auth`: Rozszerzalny system uwierzytelniania
- `django.contrib.admin`: Dynamiczny interfejs administracyjny
- `django.contrib.comments` Elastyczny system komentowania
- `django.contrib.sites` Mechanizm pozwalający jednej instancji django na uruchomienie wielu stron, każda z własną trescią i aplikacjami
- `django.contrib.sitemaps`: Narzędzie do generowania Google Sitemaps
- `django.contrib.csrf.middleware.CsrfMiddleware`: Narzędzie do zabezpieczenia przed atakami cross-site request forgery
- `django.contrib.gis`: GeoDjango - aplikacja wspierająca tworzenie rozwiązań opartych o GIS
- . . . i wiele więcej.

# Middleware

Middleware is a framework of hooks into Django's request/response processing. It's a light, low-level 'plugin' system for globally altering Django's input and/or output

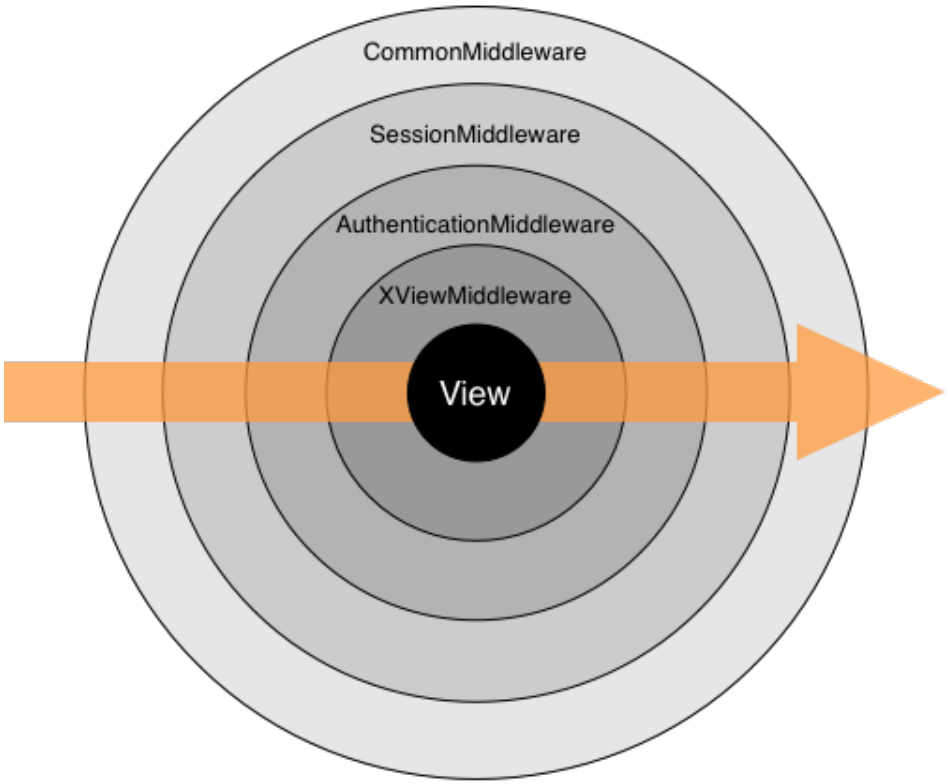
Activating middleware:

```
1  MIDDLEWARE_CLASSES = (
2      'django.middleware.common.CommonMiddleware',
3      'django.contrib.sessions.middleware.SessionMiddleware',
4      'django.contrib.auth.middleware.AuthenticationMiddleware',
5  )
```

Django applies middleware in the order it's defined in `MIDDLEWARE_CLASSES`, top-down. During the response phases the classes are applied in reverse order, from the bottom up.

- Middleware classes don't have to subclass anything.
- The middleware class can live anywhere on your Python path.

# Middleware ctd.




Middleware is like an onion: each middleware class is a "layer" that wraps the view. Each middleware component is a single Python class that defines one or more of the following methods:

- process_request
- process_view
- process_response
- process_response
- process_exception

# Ruby on Rails

# Wprowadzenie


Ruby on Rails
Rails
RoR

- framework open source do szybkiego tworzenia aplikacji webowych
- napisany w języku Ruby z użyciem architektury MVC (ang. Model-View-Controller)
- stworzony głównie przez duńskiego programistę Davida Heinemeiera Hanssona w ramach pracy nad oprogramowaniem Basecamp
- wydany jako oprogramowanie open source w lipcu 2004

# Główne założenia

- szybkość, łatwość i przyjemność pisania kodu
- reguła DRY (ang. Don't Repeat Yourself), polegająca na unikaniu powtarzania tej samej pracy w różnych miejscach
- reguła Convention Over Configuration, polegająca na sprowadzeniu do minimum niezbędnej konfiguracji przez zastępowanie jej gotowymi domyślnymi, zalecanymi wzorcami
- możliwość użycia wtyczek, które w sposób błyskawiczny rozszerzają aplikacje o rozmaite funkcje, jak logowanie wrzucanie i skalowanie obrazków, czy tagowanie

# Krótka historia

- 07-2004 pierwsze wydanie Ruby on Rails jako open source
- 08-2006 Apple zapowiada, że włączy Ruby on Rails do systemu Mac OS X v10.5 "Leopard", zrealizowane w październiku 2007
- 13-12-2024 Aktualna wersja stabilna: 8.0.1

# Instalacja

- instalacja Ruby
  ruby -v
  apt-get install ruby
- instalacja RubyGems
- instalacja Rails
  gem install rails [-v 5.1.5]

# Przykładowa aplikacja

```
1  $ rails new sample_app
2  $ cd sample_app
3  $ rails generate scaffold Post name:string title:string content:text
4  $ rails db:migrate
5  $ rails server
6  # => http://localhost:3000/posts
7
8  $ rails generate controller home index
9  $ mcedit app/views/home/index.html.erb   # your own home page
10 $ mcedit config/routes.rb   # root :to => "home#index"
```

# Komponenty Rails

- Action Pack
  Action Controller - zarządza kontrolerami
  Action View - zarządza widokami
  Action Dispatch - odpowiada za routing
- Action Mailer - framework do budowy serwisów e-mail
- Active Model - interfejs pomiedzy serwisami gem'u Action Pack gem'ami odpowiedzialnymi za mapowanie relacyjno-obiektowe
- Active Record - podstawa dla modeli
- Active Resource - framework do tworzenia web servis'ów
- Active Support - zestaw przydatnych klas i rozszerzeń standardowej biblioteki języka ruby
- Railties - główny kod Rails

# Zawartość głównego katalogu aplikacji Rails I

| File/Folder | Purpose |
|---|---|
| app/ | Contains the controllers, models, views, helpers, mailers and assets for your application. |
| bin/ | Contains the rails script that starts your app and can contain other scripts you use to deploy or run your application. |
| config/ | Configure your application's runtime rules, routes, database, and more. |
| config.ru | Rack configuration for Rack based servers used to start the application. |
| db/ | Contains your current database schema, as well as the database migrations. |
| Gemfile<br>Gemfile.lock | These files allow you to specify what gem dependencies are needed for your Rails application. |
| lib/ | Extended modules for your application. |
| log/ | Application log files. |

# Zawartość głównego katalogu aplikacji Rails II

| File/Folder | Purpose |
|---|---|
| public/ | The only folder seen to the world as-is. Contains the static files and compiled assets. |
| Rakefile | This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application. |
| README.md | This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on. |
| test/ | Unit tests, fixtures, and other test apparatus. These are covered in Testing Rails Applications |
| tmp/ | Temporary files (like cache, pid and session files) |
| vendor/ | A place for all third-party code. In a typical Rails application, this includes Ruby Gems and the Rails source code (if you optionally install it into your project). |

# Katalogi powiązane z wzorcem Model-View-Controller

| Folder | Purpose |
|---|---|
| db/migrate | migrations defining i.a. database structure |
| app/models | files with classes corresponding to tables in the database |
| app/views | html based files definig the application user interface |
| app/controllers | files with classes responsible for communication between models and views |

# MVC - Model

> rails generate model Post name:string title:string content:text

db/migrate/20100324124235_create_posts.rb

```
1  class CreatePosts < ActiveRecord::Migration
2    def self.up
3      create_table :posts do |t|
4        t.string :name
5        t.string :title
6        t.text :content
7
8        t.timestamps
9      end
10   end
11
12   def self.down
13     drop_table :posts
14   end
15 end
```

app/models/post.rb

```
1  class Post < ActiveRecord::Base
2    validates_presence_of :name, :title
3    validates_length_of :title, :minimum => 5
4    has_many :comments
5  end
```

binding models - has_one, has_many, belongs_to, ...

# MVC - Controller

## > rails generate controller posts index show edit update

app/controllers/posts_controller.rb

```
1  class PostsController < ApplicationController
2     def index
3        @posts = Post.find(:all)
4        respond_to do |format|
5           format.html # index.html.erb
6           format.xml  { render :xml => @posts }
7        end
8     end
9
10    def show
11       # ...
12    end
13    def edit
14       # ...
15    end
16    def update
17       # ...
18    end
19    ...
20 end
```

# MVC - View

app/views/posts/index.html.erb

```
1  <h1>Listing posts</h1>
2  <table>
3    <tr>
4      <th>Name</th>
5      <th>Title</th>
6      <th>Content</th>
7    </tr>
8  <% for post in @posts %>
9    <tr>
10     <td><%=h post.name %></td>
11     <td><%=h post.title %></td>
12     <td><%=h post.content %></td>
13     <td><%= link_to 'Show', post %></td>
14     <td><%= link_to 'Edit', edit_post_path(post) %></td>
15     <td><%= link_to 'Destroy', post, :confirm => 'Are you sure?', :method => :dele
16   </tr>
17 <% end %>
18 </table>
19 <br />
20 <%= link_to 'New post', new_post_path %>
```

# Konfiguracja bazy danych

- plik `config/database.yml`
- sekcje dla trzech środowisk pracy aplikacji: development, test, production
- domyślna konfiguracja bazy danych używa SQLite3
- Rails także wspiera MySQL i PostgreSQL, i posiada możliwość zastosowania plugin'ów do obsługi innych systemów baz danych
- tworzenie bazy danych: `rails db:create`

```
1  development:
2    adapter: sqlite3
3    database: db/development.sqlite3
4    pool: 5
5    timeout: 5000
```

# Lokalizacje inicjalizującego kodu

- application.rb
  RAILS_ROOT/config/application.rb
- Environment-specific Configuration Files
  RAILS_ROOT/config/environments/production.rb
  RAILS_ROOT/config/environments/development.rb
  RAILS_ROOT/config/environments/test.rb
- Initializers (load_application_initializers)
  RAILS_ROOT/config/initializers/
- After-Initializers

# Konfiguracja komponentów Rails

- configuration files: config/application.rb, config/environments/production.rb|development.rb|test.rb
- `config` object
- `config.param_name = value` - pass settings to Rails itself
- `config.component_name.param_name = value` - pass settings to individual Rails components
- `http://guides.rubyonrails.org/configuring.html` - about config parameters

```
1  config.filter_parameters += [:password]
2
3  config.active_record.timestamped_migrations = false
```

# Ustawienia środowiskowe Rails

- zmienne środowiskowe są uznawane przez różne komponenty Rails
- ENV["RAILS_ENV"] - production, development, test
- ENV["RAILS_RELATIVE_URL_ROOT"]
- ENV["RAILS_ASSET_ID"]
- ENV["RAILS_CACHE_ID"]
- ENV["RAILS_APP_VERSION"]

# Polecenia

- rails console
- rails server
- bin/rails
- rails generate
  [controller, integration_test, mailer, migration, model, observer, performance_test, plugin, resource, scaffold, session_migration]
- rails dbconsole
- rails new app_name
- rails plugin
- rails runner
- rails destroy
- rake about