Rozwiązania szkieletowe w tworzeniu aplikacji www

Tomasz Łukaszuk

Katedra Oprogramowania
Wydział Informatyki
Politechnika Białostocka

Temat wykładu:

Sesje, caching, internacjonalizacja, wdrażanie aplikacji

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.

Sesje, caching, internacjonalizacja, wdrażanie aplikacji: Agenda

1. Sesja
   - Wprowadzenie
   - Sesja w Django
   - Sesja w RoR
2. Caching
   - Wprowadzenie
   - RoR
   - Django
3. Internacjonalizacja
   - Wprowadzenie
   - Internationalisation of Django
   - Internationalisation in RoR
4. Wdrażanie aplikacji
   - Wprowadzenie
   - RoR

Wprowadzenie

Sesja - wprowadzenie

- obiekt, zapamiętujący przez pewien czas na serwerze szczegóły dotyczące połączenia z klientem
- przypisane do sesji dane mają przeważnie charakter chwilowy, ulotny (w przeciwieństwie np. do preferencji przypisywanych do konta klienta)
- sesje internetowe i sesje w obrębie pojedynczej jednostki

Sesja internetowa

- potrzeba stosowania sesji wynika z bezstanowości protokołu HTTP (niezbędny jest sposób na każdorazowe przekazywanie informacji pomiędzy przeładowaniami witryny)
- sesja oznacza uniwersalny, spersonalizowany "worek" do przechowywania danych po stronie serwera
- w sesji można przechowywać dowolne wartości, np. adres IP klienta, wybrane produkty w sklepie internetowym, informacja o zalogowaniu, identyfikatory itp.
- dla każdego klienta tworzona jest osobna sesja
- serwer musi posiadać sposób na rozpoznawanie, która sesja należy do którego klienta
- osiągane jest to poprzez stosowanie identyfikatora sesji, który po stronie klienta jest na ogół przechowywany w ciasteczku lub rzadziej w treści URL-a

Przechowywanie sesji

- po stronie serwera
  baza danych, pliki, memcache
  problemy: wydajność, wielość serwerów
- po stronie klienta
  cookies
  problemy: szyfrowanie danych, ograniczony rozmiar danych

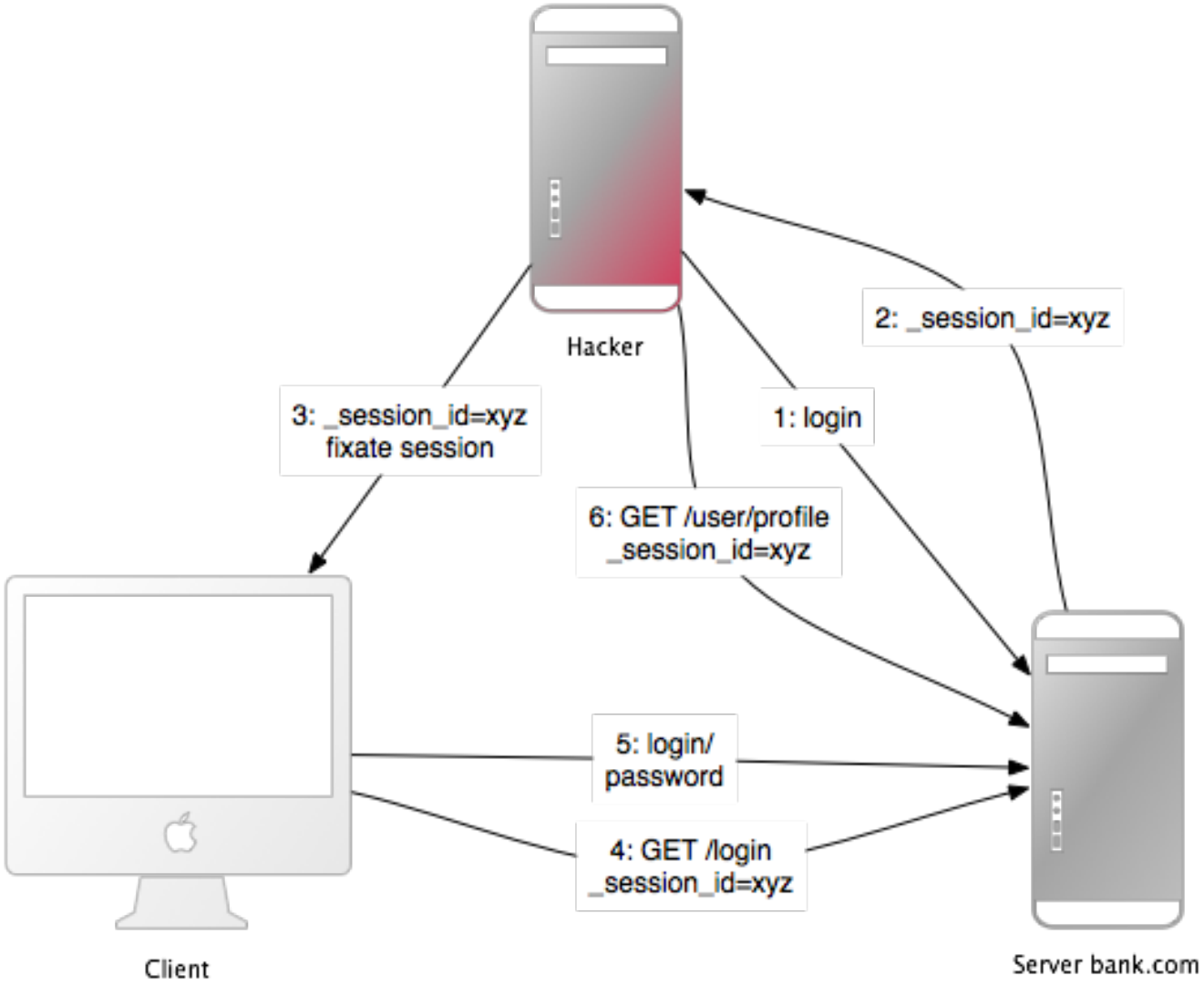
Wskazówki dotyczące sesji

- Nie przechowuj dużych obiektów w sesji. Zamiast tego powinieneś przechowywać je w bazie danych i zapisywać ich ID w sesji.
- Ważne dane nie powinny być przechowywane w sesjach. Gdy użytkownik wyczyści cookies albo zamknie przeglądarkę nie będzie mógł ich odzyskać. A wybierając metodę przechowywania sesji po stronie użytkownika, pozwolisz mu na ich odczytanie.

Przechwytywanie sesji

- odnosi się do prób przejęcia poprawnej sesji (klucza sesji), aby uzyskać nieautoryzowany dostęp do usług bądź informacji systemu komputerowego
- jest to istotne zagadnienie przy konstruowaniu stron internetowych, gdyż ciasteczka wykorzystywane do utrzymywania sesji mogą być bardzo łatwo ukradzione
- przechwycenie identyfikatora (klucza sesji), umożliwia osobie postronnej dostęp do danej strony, tak, jakby przeglądał ją pełnoprawny użytkownik

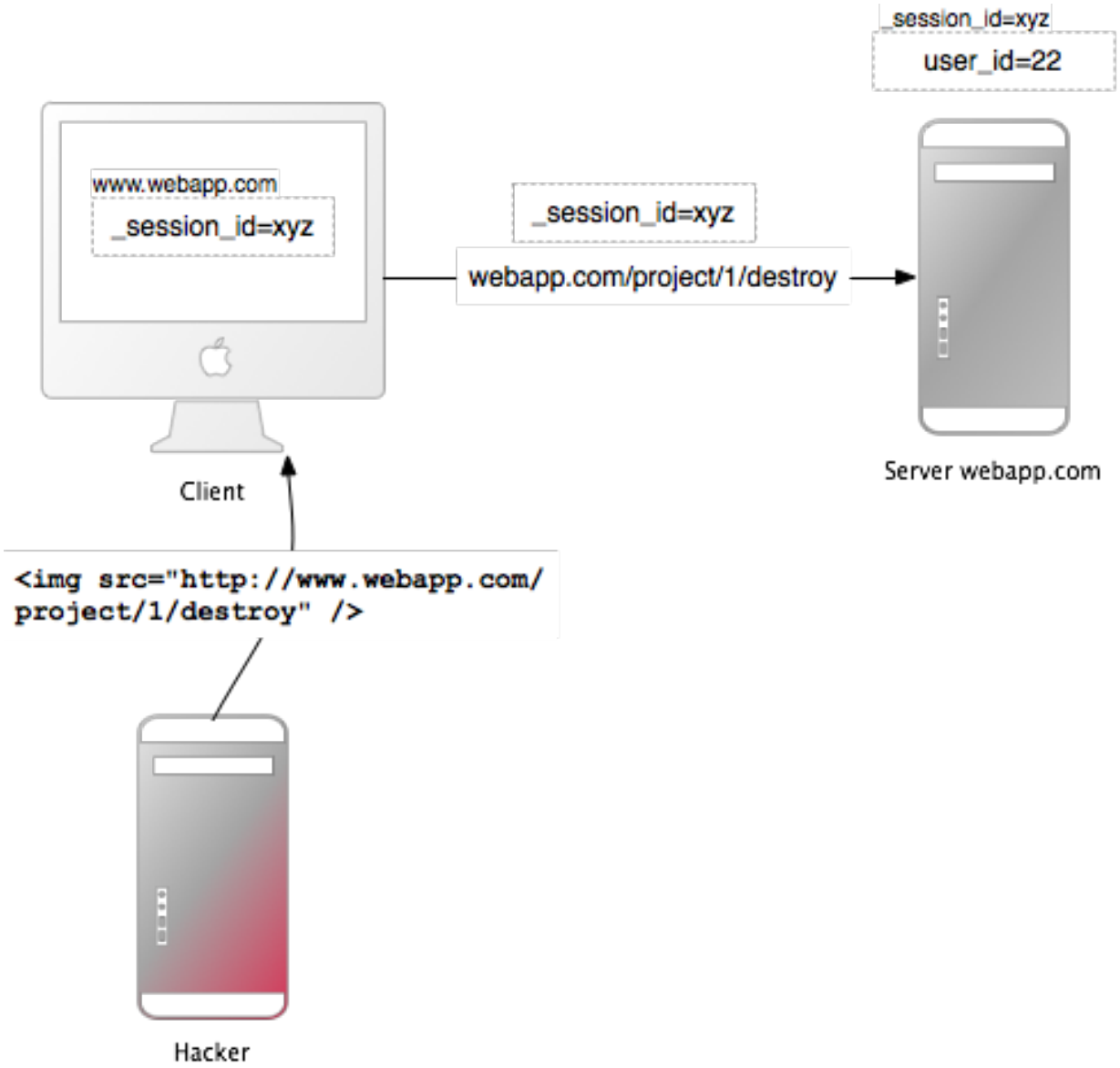
Atak typu Session fixation



Sposoby ochrony przed Session fixation

- najskuteczniejszą ochroną jest wydanie nowego identyfikatora sesji i unieważnienie starego po udanym logowaniu
- zapisywanie specyficznych właściwości użytkownika w sesji (np. adres IP, rodzaj przeglądarki), weryfikowanie ich przy każdym żądaniu i odmowa dostępu, jeśli informacje te nie pasują do siebie

Atak typu Cross-Site Reference Forgery (CSRF)



Sposoby ochrony przed CSRF

użytkownicy

- należy wylogowywać się z serwisów internetowych za każdym razem, gdy skończy się z nich korzystać oraz unikać opcji zapamiętywania sesji na komputerze
- będąc zalogowanym do serwisów internetowych, należy unikać otwierania wszelkich innych stron

twórcy aplikacji

- hasła jednorazowe
- krótki okres ważności zalogowania i czasu bezczynności
- wymóg potwierdzania i ponownego uwierzytelniania istotnych żądań
- dodawanie do każdego formularza ukrytego pola, zawierającego liczbę pseudolosową, która musi zostać przekazana wraz z żądaniem wykonania akcji, ignorowanie żądań, którym brakuje ukrytej wartości bądź gdy nie pokrywa się ona z liczbą zachowaną po stronie serwera

Sesja w Django

Włączanie/Wyłączanie obsługi sesji

- Sesje są implementowane jako element middleware.
- Aby włączyć funkcjonalność sesji w swoim projekcie należy:
  1) do ustawień `MIDDLEWARE_CLASSES` dodać `django.contrib.sessions.middleware.SessionMiddlewar`
  2) do ustawień `INSTALLED_APPS` dodać `django.contrib.sessions`
- Domyślnie `settings.py` stworzony przez `django-admin startproject` zawiera powyższe ustawienia

Konfiguracja sesji

- Domyślnie django przechowuje dane sesji w bazie danych (model `django.contrib.sessions.models.Session`)
- Ustawienie `SESSION_ENGINE` determinuje sposób przechowywania sesji
- `SESSION_ENGINE=django.contrib.sessions.backends.cache` - sesje przechowywane w cache
- `SESSION_ENGINE=django.contrib.sessions.backends.cache_db` - sesje przechowywane w cache i bazie danych
- `SESSION_ENGINE=django.contrib.sessions.backends.file` - sesje przechowywane w plikach po stronie serwera
- `SESSION_ENGINE=django.contrib.sessions.backends.signed_cookies` - sesje przechowywane w plikach cookie

Używanie sesji w funkcjach widoków

- Jeżeli `SessionMiddleware` jest aktywowany to każdy obiekt `HttpRequest` (pierwszy argument każdej funkcji widoku) zawiera słownikopodobny obiekt `session`.
- Do `request.session` można pisać i czytać w dowolnym punkcie widoku. Można go edytować wiele razy.

```
1  fav_color = request.session['fav_color']
2  request.session['fav_color'] = 'blue'
3  del request.session['fav_color']
4  'fav_color' in request.session
5  fav_color = request.session.get('fav_color', 'red')
6  fav_color = request.session.pop('fav_color')
```

Sesja w RoR

- As in Django in RoR session is dictionary-like structure allowing for storing data for particular client between requests
- Each session is identified by session_id
- session_id is 32 characters generated by MD5 based
  - time
  - constant string
  - random value
- Session is not full dictionary (hash) — not all methods work

Session management

- Session can be disabled in particular controllers
- This can save memory and processing time
- We can enable session for some controllers
- Or only for only chosen actions

```
1  class MyController < ActionController::Base
2      session :off
3  end
4
5  class OtherController < ActionController::Base
6      session :on, :only => [:create, :update]
7  end
```

Session usage

- Session behaves as an hash
- Keys are strings or constants
- Values can be any type
- If value is not needed it can be removed using “delete” method
- Setting value to “nil” also causes deleting from session
- reset_session() removes all object from session object

```
1  session[:key] = value
2
3  session[:name] = 'Rybak'
4  session[:email] = 'rybak@example.com'
5  session[:messages] = ['Line1', 'Line2']
6
7  session.delete :messages
```

Session storage

- Session values for popular site can take much space
- Management of large amount of data can be challenging
- Proper management may mean difference between working site and site that users must wait for any action
- Ruby on Rails offers four different ways of storing session data
- It is chosen for entire application in "config.action_controller.session_store"

CookieStore stores data on client; cookie size limit of 4kB applies

DRBStore

MemCacheStore

ActiveRecordStore session stored in database

```
1  config.action_controller.session_store = :drb_store
2  config.action_controller.session_store = :active_record_store
```

Flash session

- Variable called “flash”
- It is special part of session
- Like “session” it is hash storing key/value pairs
- Values stored in “flash” are cleared after next request
- To make them available in the current session (e.g. to use flash values in rendering of other action) use “flash.now”
- flash.keep causes values to be persisted to use in next requests
- This function accepts optional list of names to persist

```
1  flash.now[:key] = value
2  flash.keep(:kay)
```

Usage of cookies in sessions

- All sessions store session_id in the cookie
- ActionController::SessionManagement
- Data stores in the cookie store is signed (so tampering is very hard) but is not encrypted, so anyone can read it

Cookies

- As noted earlier, cookies are stored on the client
- Cookie size limit is 4kB
- Client need not to store or return cookies
- Variable “cookies” behaves as an hash, but does not support all dictionary methods
- It is similar to variable “session” in this regard

```
1  cookies[:key] = value
2
3  cookies[:name] = 'Rybak'
4
5  cookies.delete(:messages)
```

Cookie creation

- When creating cookies it is possible to set parameters managing their lifetime options

value	
path	default “/”
domain	
expires	
secure	whether to send cookie only over HTTPS

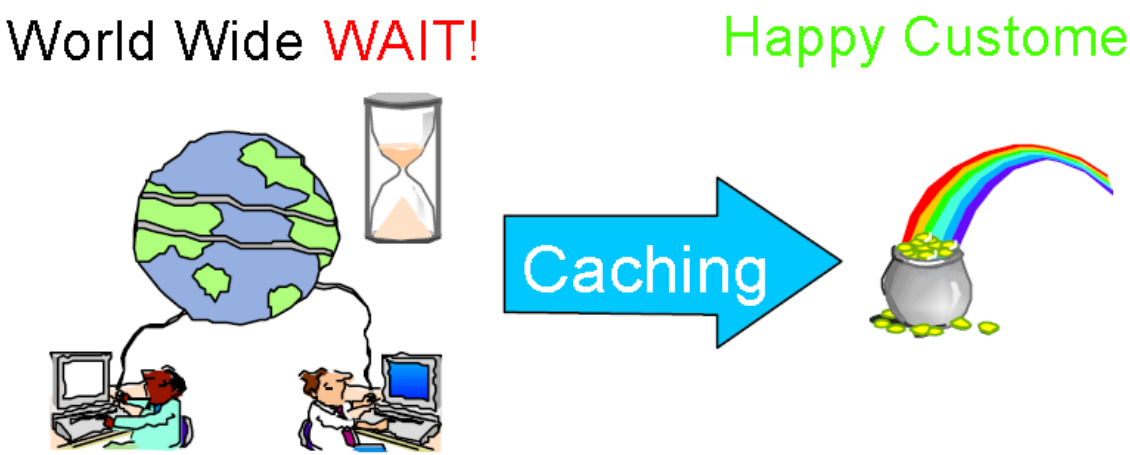
```
1   cookies[:name] = {:value => 'Rybak', :path = '/administration'}
```

Caching

Cache

- cache w informatyce - pamięć podręczna; mechanizm, w którym ostatnio pobierane dane dostępne ze źródła o wysokiej latencji i niższej przepustowości są przechowywane w pamięci o lepszych parametrach, CPU cache, Disk cache
- web cache
  - cache przeglądarki, proxy servers cache - zapamiętuje wcześniejsze odpowiedzi serwera, tj. strony, fragmenty stron; redukuje ilość informacji przesyłanych siecią
  - cache aplikacji sieciowych - zapamiętuje kopie dokumentów; jeżeli spełnione są określone warunki odpowiedź serwera generowana jest na podstawie zawartości cache'u

Cache



source: www.almaden.ibm.com/u/mohan/Caching_VLDB2001.pdf

Caching w działaniu

```
1  if (strona jest w cacheu) then
2      return strona z cacheu
3  else {
4      wygeneruj stronę
5      zapisz wygenerowaną stronę do cacheu (do następnego użycia)
6      return wygenerowana strona
7  }
```

Cache - zagadnienia

- autoryzacja do zapamiętanych w cache stron (różne metody cache'owania)
- odświeżanie, aktualność cache
- cache stores: pamięć, pliki, memcache

Ruby on Rails

Caching w Ruby on Rails

- Cache'owanie stron, akcji i fragmentów
- Sweepers
- sposoby przechowywania cache'u – Cache Store
- wsparcie dla warunkowego GET
- zaawansowane mechanizmy (dodatkowe plugin'y)

Ustawienia caching'gu

```
1  config.cache_classes = true | false
2
3  config.action_controller.perform_caching = true | false
4
5  config.cache_store =
6       :memory_store | :file_store | :drb_store
7       :mem_cache_store | :synchronized_memory_store |
8          :compressed_mem_cache_store
9       (custom store)
10
11 config.action_controller.page_cache_directory
12 config.action_controller.page_cache_extension
```

Cache'owanie strony

- tworzenie statycznych stron będących wynikami przetwarzania akcji
- jest super szybki
- nie może być stosowany w każdej sytuacji (autentykacja)
- o aktualność cache'u musi zadbać programista
- ignoruje wszystkie parametry wywołania (/products?page=1 => products.html)

```
1  class ProductsController < ActionController
2
3     caches_page :index
4
5     def index
6        @products = Products.all
7     end
8
9     def create
10       expire_page :action => :index
11    end
12
13 end
```

Cache'owanie akcji

- pozwala na autentykację i inne restrykcje dotyczące uruchamiania
- before filters będą uruchamiane przed zwróceniem zawartości cache'u
- czyszczenie cache'u działa w ten sam sposób jak w przypadku page caching

```
1  class ProductsController < ActionController
2
3     before_filter :authenticate
4     caches_action :index
5
6     def index
7        @products = Products.all
8     end
9
10    def create
11       expire_page :action => :index
12    end
13
14 end
```

Cache'owanie fragmentów

- pozwala na opakowanie fragmentu widoku w cache block i ładowanie go z cache'u przy następnym żądaniu

```
1  <% Order.find_recent.each do  |o| %>
2    <%= o.buyer.name %> bought <% o.product.name %>
3  <% end %>
4
5  <% cache do %>
6    All available products:
7    <% Product.all.each do |p| %>
8      <%= link_to p.name, product_url(p) %>
9    <% end %>
10 <% end %>
```

Cache'owanie fragmentów II

- zarządzanie cache'owaniem fragmentów

```
1  #view
2  <% cache(:action => 'recent', :action_suffix => 'all_products') do %>
3     ...
4  <% end %>
5
6  #controller
7  expire_fragment(:controller => 'products', :action => 'recent',
8                  :action_suffix => 'all_products')
```

```
1  #view
2  <% cache('all_available_products') do %>
3    ...
4  <% end %>
5
6  #controller
7  expire_fragment('all_available_products')
```

Sweepers

- przeniesienie wszystkich potrzebnych działań do odświeżania zawartości cache'u do podklasy ActionController::Caching::Sweeper (obserwator)

```
1  class ProductSweeper < ActionController::Caching::Sweeper
2     observe Product
3
4     def after_create(product)
5        expire_cache_for(product)
6     end
7     def after_update(product)
8        expire_cache_for(product)
9     end
10    def after_destroy(product)
11       expire_cache_for(product)
12    end
13
14    private
15    def expire_cache_for(product)
16       expire_page(:controller => 'products', :action => 'index')
17       expire_fragment('all_available_products')
18    end
19 end
```

Sposoby przechowywania cache'u – Cache Store I

- cache'owanie stron - zawsze przechowywane na dysku
- różne magazyny do obsługi cache'u dla mechanizów cache'owania akcji i fragmentów

1. ActiveSupport::Cache::MemoryStore
   - przechowuje wszystko w pamięci w tym samym procesie
   - dla aplikacji nie wykonujących wygaśnięcia pozycji cache'u
   - jest zdolny do przechowywania nie tylko łańcuchów znaków, ale i dowolnych obiektów Ruby
   - nie jest bezpieczny-wątkowy
2. ActiveSupport::Cache::FileStore
   - dane są przechowywane na dysku
   - domyślny sposób przechowywania cache'u, domyślna ścieżka to tmp/cache
   - pracuje dla wszystkich typów środowisk

Sposoby przechowywania cache'u – Cache Store II

3. ActiveSupport::Cache::DRbStore
   - dane są przechowywane w oddzielnych procesach DRb, które komunikują się z wszystkimi serwerami
   - wymaga uruchomienia i zarządzania na oddzielnych procesach DRb
4. ActiveSupport::Cache::MemCacheStore
   - działa podobnie do DRbStore, ale zamiast niego używa memcached Dangi
   - obecnie najbardziej popularny sposób przechpwywania cache'u dla produkcji stron www
5. ActiveSupport::Cache::SynchronizedMemoryStore
6. ActiveSupport::Cache::CompressedMemCacheStore

Wsparcie dla warunkowego GETa

- warunkowe GETy są elementami specyfikacji protokołu HTTP
- webserwer może poinformować przeglądarkę, że odpowiedź na żądanie GET nie została zmieniona od czasu ostatniego żądania i może być bezpiecznie wyciągnięta z pamięci cache przeglądarki
- HTTP_IF_NONE_MATCH, HTTP_IF_MODIFIED_SINCE

```
1  class ProductsController < ApplicationController
2    def show
3      @product = Product.find(params[:id])
4      if stale?(:last_modified => @product.updated_at.utc, :etag => @product)
5        # ... normal response processing
6      end
7    end
8  end
```

```
1  class ProductsController < ApplicationController
2    def show
3      @product = Product.find(params[:id])
4      fresh_when :last_modified=>@product.published_at.utc, :etag=>@product
5    end
6  end
```

Cache'owanie niskopoziomowe

W cache'u można zapisywać dowolne obiekty Ruby.

```
1  Rails.cache.read("city")    # => nil
2  Rails.cache.write("city", "Duckburgh")
3  Rails.cache.read("city")    # => "Duckburgh"
```

Django

Konfiguracja cache'u I

- Określenie gdzie (w jaki sposób) mają być przechowywane dane zapisywane do cache: baza danych, system plików, bezpośrednio w pamięci
- Ustawienie `CACHES`
- Memcached

```
1  CACHES = {
2      'default': {
3          'BACKEND': 'django.core.cache.backends.memcached.PyMemcacheCache',
4          'LOCATION': '127.0.0.1:11211',
5      }
6  }
```

- Database caching

Konfiguracja cache'u II

```
$ python manage.py createcachetable
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'my_cache_table',
    }
}
```

- Filesystem caching

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.filebased.FileBasedCache',
        'LOCATION': '/var/tmp/django_cache',
    }
}
```

- Local-memory caching

Konfiguracja cache'u III

```
1  CACHES = {
2      'default': {
3          'BACKEND': 'django.core.cache.backends.locmem.LocMemCache',
4          'LOCATION': 'unique-snowflake',
5      }
6  }
```

- Dummy caching

```
1  CACHES = {
2      'default': {
3          'BACKEND': 'django.core.cache.backends.dummy.DummyCache',
4      }
5  }
```

Cache'owanie stron

- najprostsze rozwiązanie
- cache'owanie całych stron
- cache'owane są wszystkie odpowiedzi GET i HEAD ze statusem 200
- aby aktywować dodaj 'django.middleware.cache.UpdateCacheMiddleware' i 'django.middleware.cache.FetchFromCacheMiddleware' do MIDDLEWARE_CLASSES
- dodaj ustawienie w pliku settings.py
  CACHE_MIDDLEWARE_ALIAS
  CACHE_MIDDLEWARE_SECONDS
  CACHE_MIDDLEWARE_KEY_PREFIX

Cache'owanie na poziomie funkcji widoków

- bardziej szczegółowy sposób na wykorzystanie mechanizmu cache'owania
- buforuje wyjście wskazanych funkcji widoków
- dekorator cache_page

```
1  from django.views.decorators.cache import cache_page
2
3  @cache_page(60 * 15)
4  def my_view(request, param):
5      # ...
```

```
1  from django.views.decorators.cache import cache_page
2
3  urlpatterns = [
4      path('foo/<int:code>/', cache_page(60 * 15)(my_view)),
5  ]
```

Cache'owania fragmentów szablonów

- buforuje fragmenty szablonów
- `cache` template tag

```
1   {% load cache %}
2   {% cache 500 sidebar %}
3       .. sidebar ..
4   {% endcache %}
```

buforowanie wielokrotnych kopi fragmentu w zależności od pewnych danych dynamicznych, które pojawiają się wewnątrz fragmentu

```
1   {% load cache %}
2   {% cache 500 sidebar request.user.username %}
3       .. sidebar for logged in user ..
4   {% endcache %}
```

Niskopoziomowe cache API

- moduł django.core.cache
- pozwala przechowywać w pamięci podręcznej obiekty z dowolnego poziomu szczegółowości (strings, dictionaries, lists of model objects, etc.)
- podstawowy interfejs set(key, value, timeout_seconds) i get(key):

```
1  >>> cache.set('my_key', 'hello, world!', 30)
2  >>> cache.get('my_key')
3  'hello, world!'
4
5  >>> cache.set('a', 1)
6  >>> cache.set('b', 2)
7  >>> cache.set('c', 3)
8  >>> cache.get_many(['a', 'b', 'c'])
9  {'a': 1, 'b': 2, 'c': 3}
10
11 >>> cache.delete('a')
```

Internacjonalizacja

Wprowadzenie

i18n and l10n

- internationalisation — i18n
- Extending programs to allow to run them different environments
- localisation — l10n
- Implementing specific language for internationalised software
- To achieve l10n we need to take care about i18n first

Need for localisation and internationalisation

- Most of software created in English
- Computers (languages, programs) come from English-speaking countries
- Most of programmers (should) speak English
- But ordinary users do not!
- Also necessity to “speak” in culture of country
- especially programs used in business, offices, etc.

Problems with localisation

- Different encodings of national symbols
- Direction of writing (left-to-right or right-to-left)
- Different lengths of strings
- Various modes of plurality
- Different calendars and time systems (including time zones, daylight saving rules, etc.)
- Displaying numbers
- Currency

Linux environment variables

LANG	
LC_ALL	meta-variable setting all features
LC_CTYPE	character encoding
LC_COLLATE	sorting, etc.
LC_MESSAGES	
LC_MONETARY	
LC_NUMERIC	
LC_TIME	

Translating program messages

- Surround all strings with gettext call
  - Usually name shorter than “gettext” is defined
- Use xgettext to analyse sources of program and extract translatable strings
- Use msginit to create human-readable file from generated text
- Translate
- Use msgfmt to create machine-readable file containing translated strings

```
1  #define _(x) gettext(x)
2
3  printf(_("Hello world.\n");
```

- POT file contains template of all strings to translate
- It serves as a base for all languages
- There is only one such file for each program
- PO file contains original and translated strings
- One PO file exists for each of languages
- MO file is used by gettext to provide program with translated strings

msgid	identifier of string
msgstr	translated string
fuzzy	
c-style	contains C language formatting

Internationalisation of Django

Language information from HTTP

Accept-Language

Accept-Charset

- User-agent can use those headers to inform server about understood languages and their preferences
- Accept-Language: pl, en; q=0.9, de; q=0.4, fr; q=0.1
- It allows for server to send response in language best suiting the user
- Of course user does no build Accept-Language header
- Browser builds this header based on configuration and environment

Localising Django applications

- Django is Python application
- Python under Unix can use gettext
- Django uses gettext via Python gettext module
- Instead of name gettext, underscore is used to save typing
  - In C preprocessor is used
  - In Python one can use aliasing during import
- In some environments (Solaris, wx)"_T" is used

```
1  from django.utils.translation import gettext as _
2
3  string = _("Hello world!")
```

Usage of gettext

- Module django.utils.translation
- Function gettext

```
1  from django.utils.translation import gettext
2
3  def view(request):
4          output = ugettext("Hello world!")
5          return HttpResponse(output)
```

Gettext functions in Django

gettext	ordinary function
ngettext	translation function taking pluralisation into consideration
gettext_noop	function returning ordinary string, used during preparation to internationalisation
gettext_lazy	lazy translation; returns object which translates message only when string is really needed

Lazy translation

- When using lazy translation message will be translated as late as possible
- It means that different thread might create string and different use it
- This situation can occur in model classes
- Also localisation settings are correctly set late, just before calling view function

```
1  from django.utils.translation import gettext_lazy as _
2
3  class MyModel(models.Model):
4      class Meta:
5          verbose_name = _("name for admin")
6          verbose_name_plural = _("names for admin")
```

Strings in Python

- Two types — ordinary and Unicode
- When two strings of different types are mixed, they are “casted” to Unicode
- Unicode and ordinary strings should not be mixed in i18n
- They are not compatible, especially with lazy translating
- Also many Python functions accept string and will behave strangely when given lazy string
- allow_lazy decorator
- from django.utils.functional import allow_lazy
- It halts execution of function until its result is needed
- It can be used for functions accepting and returning strings (both Unicode and ordinary)

Translating templates

- Tag trans
- One can add option like noop at the end

```
1   <h1>{\% trans "This is header" \%}</h1>
```

- Short texts can be treated as one entity
- Longer texts are harder to manage
  - are longer
  - can contain different sections and values from variables
- Block translation
- Tag blocktrans
- Ability to use filters and naming values coming from such filtering
- Tag “plural” separates single and plural case

```
1  <p>
2  {\% blocktrans with value|filter as name \%}
3  This is some {{ name }} that should be translated
4  {\% endblock \%}
5  </p>
```

RequestContext variables

get_available_languages function returning set of available languages

get_current_language function returning language preferred by current user

get_current_language_bidi function returning direction of text

RequestContext extension describing following variables

LANGUAGES collection of pairs language symbol–language name

LANGUAGE_CODE current user preferred language; added to HttpReqest variable by middleware

LANGUAGE_BIDI direction of language (left-to-right if True)

Creation of language files

- Requires gettext installation
- `django-admin makemessages`
- Scans all source files and extracts strings marked for translation
- “-l LOCALE” choses locales to be created
- “-a” rescans all sources for new or changed strings
- “-e EXT” extension of files to scan
- “-i EXT” extension of files to ignore

Managing translation files

- File created in locale/LANG/LC_MESSAGES/django.po
- Directory “locale/” must exist
- It is not created by the tool
- `django-admin compilemessages`
- It creates .mo files from .po files
- It should be run after all changes in translations

Enabling internationalisation

- Package django.middleware.locale.LocaleMiddleware
- Add module to MIDDLEWARE_CLASSES
- It should be placed after Session and Cache middleware and before other modules
- LANGUAGE_CODE settings variable sets default language settings for Django

Available languages

- LANGUAGES variable
- Contains collection of pairs (language code, language name)
- Name of language should be matched as translated
- I.e. surrounded by gettext function variant
- But use dummy translation function (e.g. gettext = lambda s: s)
- Only languages present in this variable will be taken into consideration

Order of looking for translated messages

1. Application directory
2. Project directory
3. All directories from LOCALE_PATHS
4. Django installation

Order of checking for language to use

1. Check django_language key in session
2. Look for cookie
   - default name “django_language”
   - can be changed by setting LANGUAGE_COOKIE_NAME
3. Look at Accept-Language header from HTTP
4. Value of LANGUAGE_CODE variable

Setting language by user in the browser

- Sometimes we want to allow user to change displayed language
- Browser headers are not always reliable

```
1  urls.py:
2  path('i18n/', include('django.conf.urls.i18n')),
```

- It will enable URL i18n/setlang
- After setting language by user browser is redirected to previous page
- . . . or root if there is no previous page
- View django.views.i18n.set_language
- It expects to be called via POST with parameter “language”
- It will set variable in session or set cookie

Internationalisation in RoR

Internationalisation API

I18n	module containing methods to deal with i18n
translate	- like gettext
t	short alias to function “translate”; can be used in form I18n.t ’string’
localize	function localising data and time objects
l	alias to function localize
load_path	path to files for localisation
locale	currently used locale
default_locale	

Localisation files

- Directory config/locales contains files with translated messages
  - en.yml
  - pl.yml
  - LANGUAGE.yml
- File name should contain only language part (pl) not region part (de-AT)
- I18n.load_path contains array of paths to files to load

Sample files with translated messages

```
1  en:
2      hello: Hello world!
3      weather: Nice weather, isn't it?
4
5  pl:
6      hello: Serdecznie witamy!
7      weather: Jaka piekna pogoda!
```

Setting locale

- RoR does not use one method of choosing locale used for particular user
- There are many methods to do so
  - Use cookies (not recommended)
  - Use session (not recommended)
  - Store parameter in URL
  - Check country IP belongs to

```
1  before_filter :set_locale
2  def set_locale
3      I18n.locale = params[:locale] || I18n.default_locale
4  end
```

Locale setting basing on domain from URL

```
1  before_filter :set_locale
2  def set_locale
3    I18n.locale = extract_locale_from_uri || I18n.default_locale
4  end
5  # You have to put something like:
6  #   127.0.0.1 application.com
7  #   127.0.0.1 application.it
8  #   127.0.0.1 application.pl
9  # in your /etc/hosts file to try this out locally
10 def extract_locale_from_tld
11   parsed_locale = request.host.split('.').last
12   (available_locales.include? parsed_locale) ? parsed_locale  : nil
13 end
```

Various utility functions

available_locales variable storing array of strings with symbols of present localisations

load_translations(file) function for loading translated files

translations dictionary with all available locales

translations.keys

Localising time displaying

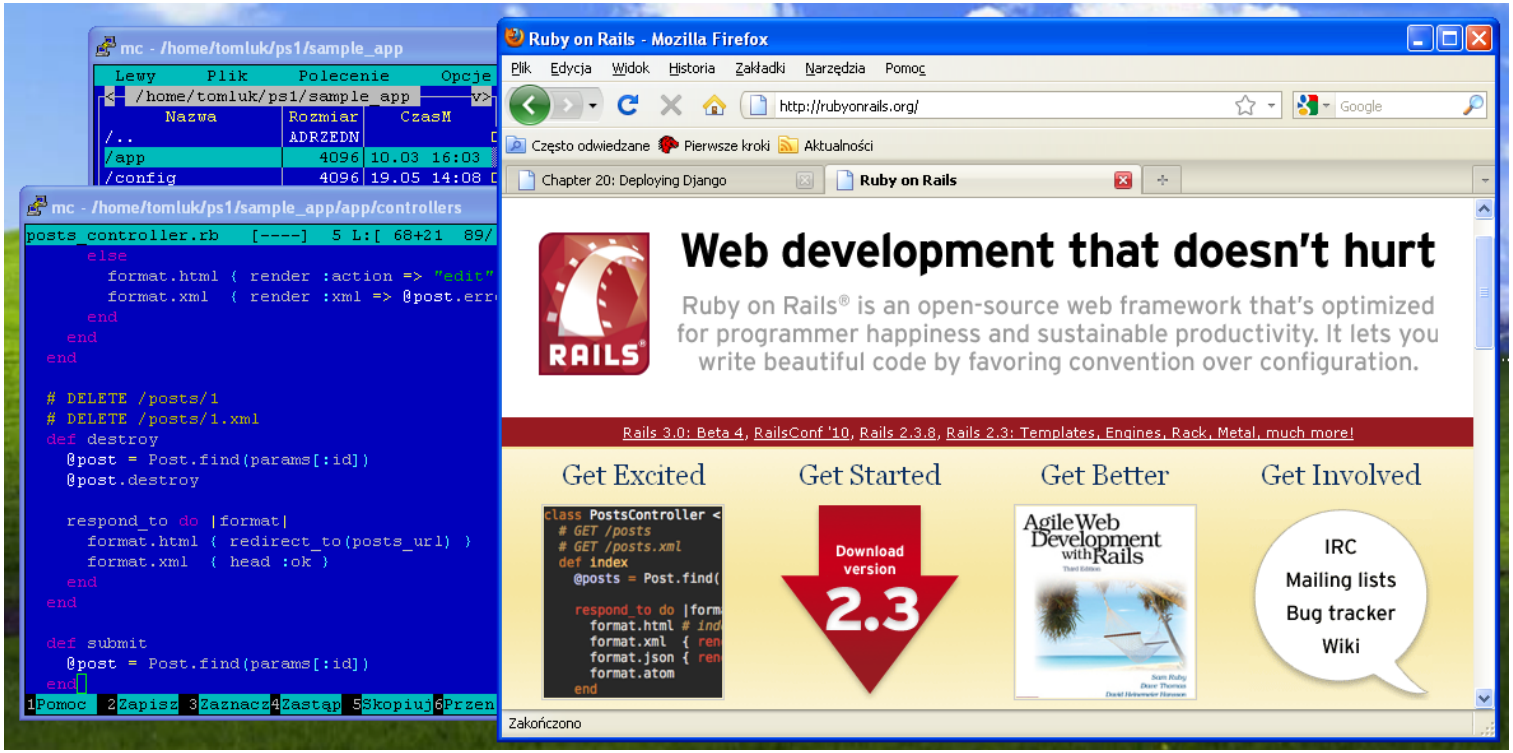
- In Unix time is stored as
  - number of seconds after 1970-01-01
  - in structure “struct tm”
- Function strftime can be used to print time
- It accepts string with special symbols for years, hours, minutes, days of week, etc.
- I18n.l function from RoR takes localised time format string and uses it to print datetime in appropriate format

```
1  pl:
2      time:
3          formats:
4              short: "Jest godzina %H"
5
6  l Time.now, :format => :short
```

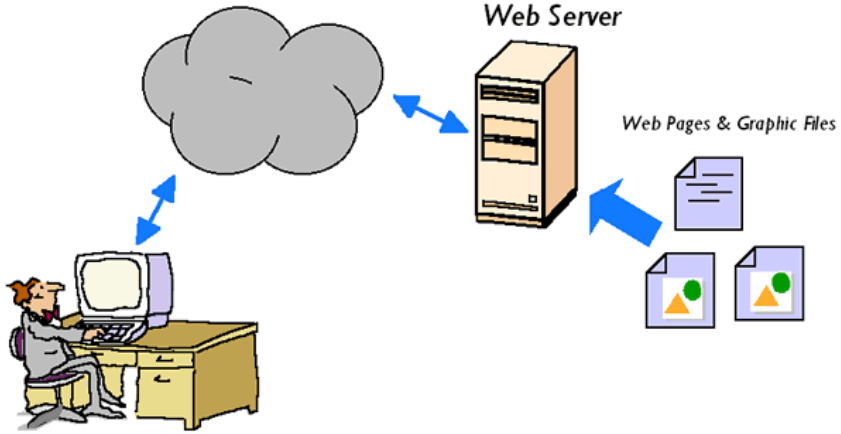
Wdrażanie aplikacji

Wdrażanie

- Wdrażanie oprogramowania obejmuje wszystkie działania prowadzące do udostępnienia oprogramowania użytkownikom.
- Wdrażanie aplikacji web'owej - udostępnienie jej w Internecie



Serwer WWW



- program działający na serwerze internetowym, obsługujący żądania protokołu komunikacyjnego HTTP
- z serwerem WWW łączy się, poprzez sieć komputerową, przeglądarka internetowa, będąca jego klientem, aby pobrać wskazaną stronę WWW
- podstawowym zadaniem serwera WWW jest dostarczanie stron na żądanie klientów korzystających z protokołu HTTP, oznacza to dostarczanie dokumentów HTML i dodatkowych treści, które mogą być zawarte w dokumencie, takich jak obrazy, arkusze stylów i skrypty

Ruby on Rails

Serwery www

1. WEBrick
   - szybki i prosty serwer aby rozpocząć budowę aplikacji
   - napisany w Ruby
   - nie polecany do pracy na etapie produkcyjnym
   - instalowany razem ze środowiskiem ruby
   - uruchamiany w starszych wersjach Rails poleceniem `rails server`
2. Mongrel
   - mała biblioteka dostarczająca bardzo szybkiego serwera
   - wspiera budowanie klastrów (Loadalancing)
   - release 1.1.5 is no longer able to install in Ruby versions higher than 1.9.2
3. Phusion Passenger
   - używa mod_rails lub mod_rack do współpracy z Apache lub Nginx
   - wspiera Apache oraz szybki i lekki serwer www Nginx
   - http://www.modrails.com/

Phusion Passenger i Apache I

- instalacja

```
1  gem install passenger
2  passenger-install-apache2-module
```

Listing 1: /etc/apache2/mods-enabled/passenger.conf

```
1  LoadModule passenger_module \
2    /usr/lib/ruby/gems/1.8/gems/passenger-2.2.11/ext/apache2/mod_passenger.so
3  PassengerRoot /usr/lib/ruby/gems/1.8/gems/passenger-2.2.11
4  PassengerRuby /usr/bin/ruby1.8
```

- wdrażanie aplikacji Ruby on Rails
  /webapps/rails/myapp => www.myhost.pl/myrailssite

Phusion Passenger i Apache II

Listing 2: apache.conf current virtual host

```
1  <VirtualHost *:80>
2      ServerName www.myhost.pl
3      DocumentRoot /var/websites
4      <Directory /var/websites>
5          Allow from all
6      </Directory>
7  </VirtualHost>
```

Listing 3: make a symlink

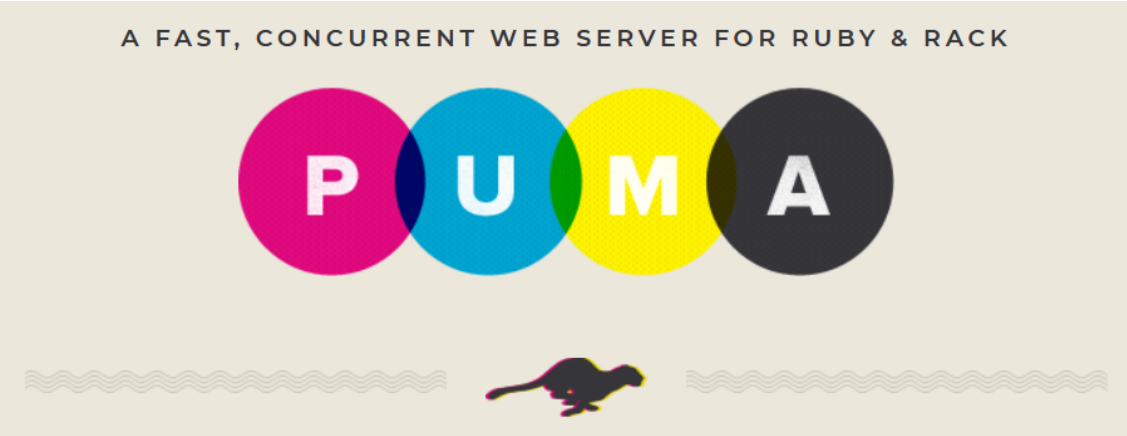
```
1  ln -s /webapps/rails/myapp/public var/websites/myrailssite
```

Phusion Passenger i Apache III

Listing 4: apache.conf

```
1  <VirtualHost *:80>
2      ServerName www.myhost.pl
3      DocumentRoot /var/websites
4      <Directory /var/websites>
5          Allow from all
6      </Directory>
7  </VirtualHost>
8
9  RailsBaseURI /myrailssite
10 <Directory /var/websites/myrailssite>
11     Options -MultiViews
12 </Directory>
```

Puma web server



Unlike other Ruby Webservers, Puma was built for speed and parallelism. Puma is a small library that provides a very fast and concurrent HTTP 1.1 server for Ruby web applications. It is designed for running Rack apps only.

Hosting Rails

- cal.pl
- hostingrails.pl
- newrails.pl
- bluehost.com
- hostmonster.com
- lunarpages.com
- railsplayground.com
- hostingrails.com
- www.alwaysdata.com

Django

WSGI

- Web Server Gateway Interface
- Pythonowy standard komunikacji dla aplikacji HTTP
- definiuje prosty i uniwersalny interfejs pomiędzy serwerami WWW a aplikacjami internetowmi lub frameworkami języka programowania Python
- Apache mod_wsgi
- dwie strony WSGI: strona serwera i strona aplikacji, serwer wywołuje funkcje aplikacji, dostarcza informacji środowiskowej i otrzymuje treść stron internetowych w zamian

WSGI i django

- Apache mod_wsgi
- httpd.conf

```
1  WSGIScriptAlias / /path/to/mysite.com/mysite/wsgi.py
2  WSGIPythonPath /path/to/mysite.com
3
4  <Directory /path/to/mysite.com/mysite>
5  <Files wsgi.py>
6  Order deny,allow
7  Require all granted
8  </Files>
9  </Directory>
```

Django hosting

- pythonanywhere.com
- linuxpl.com
- www.alwaysdata.com
- djangohosting.ch
- dreamhost.com
- http://code.djangoproject.com/wiki/DjangoFriendlyWebHosts