Open Source Frameworks for Rapid Application Development

Marek Krętowski
Krzysztof Bandurski, Tomasz Łukaszuk, Tomasz Rybak

Software Departament
Faculty of Computer Science
Bialystok University of Technology

m.kretowski@pb.edu.pl
k.bandurski@pb.edu.pl, t.lukaszuk@pb.edu.pl, t.rybak@pb.edu.pl

Lecture topic

Configuration and conventions in Django

Configuration and conventions in Django: Table of content

Introduction

General

django

- A high-level Python Web framework that encourages rapid development and clean, pragmatic design
- Follows the Model-View-Controller (MVC) architectural pattern, but Django's "view" corresponds to controller, whereas Django's "template" corresponds to view - read more in Django FAQ.
- Originally developed to manage several news-oriented sites for The World Company of Lawrence, Kansas
- Released publicly under a BSD license in July 2005
- Named after gypsy jazz guitarist Django Reinhardt
- Since June 2008 managed by Django Software Foundation
- Latest release: 1.3.1

Components

- A lightweight, standalone web server for development and testing
- A form serialization and validation system.
- A caching framework which can use any of several cache methods.
- Support for middleware classes which can intervene at various stages of request processing.
- An internal dispatcher system which allows components of an application to communicate events to each other via pre-defined signals.
- An internationalization system.
- A serialization system which can produce and read XML and/or JSON representations of Django model instances.
- A system for extending the capabilities of the template engine.
- An interface to Python's built-in unit test framework.

Loose coupling

Loose coupling means that individual components of Django's feature stack are kept as separate as possible. For example:

- Django templating language makes it very pleasant to represent the contents of your models, but its equally easy to use Django's templating language to represent other kinds of Python objects as well.
- If you don't like Django's templating language, you can use replace it with e.g. Jinja2
- The Django ORM makes it easy to setup and access a database, but it possible to use SQLAlchemy instead.

If you prefer to use tools other than those provided with Django, you can use them.

While Django will let you make decisions, it will rarely make you make a decision.

Installation

- install Python
- install a database server (PostgreSQL, MySQL, Oracle or SQLite) and relevant Python bindings
- install Django
  - from source: download and extract Django-NNN.tar.gz, then `setup.py install`
  - using apt (Linux): `apt-get install Django`
  - using yum (Linux): `yum install Django`

Starting a project

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

- A project is a directory (here: `mysite`) containing a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings.
- A project can use any number of Django applications.
- Contents of `mysite`:
  - `__init__.py`: An empty file that tells Python that this directory should be considered a Python package.
  - `manage.py`: A command-line utility that lets you interact with this Django project in various ways.
  - `settings.py`: Settings/configuration for this Django project.
  - `urls.py`: The URL declarations for this Django project; a "table of contents" of your Django-powered site.

Project files

Administration: `django-admin.py` and `manage.py`

Usage:

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

- `django-admin.py` is Django's command-line utility for administrative tasks.
- `manage.py` is a thin wrapper around `django-admin.py`
  - puts your project's package on sys.path.
  - sets the DJANGO_SETTINGS_MODULE environment variable so that it points to your project's settings.py file.
- *subcommand* tells the script what to do. Examples:
  - `syncdb`: Creates the database tables for all apps used in the project.
  - `shell`: Sets up the the project's environment and starts the Python interactive interpreter
- Applications can register their own actions with `manage.py`
- When working on a single Django project, it's easier to use `manage.py`

Project settings: `settings.py`

`settings.py` is a Python module that contains all the configuration of your Django project.
Using settings in Python code:

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

Important notes:

- Always use settings by importing the object `django.conf.settings`
- Do not alter settings at runtime
- ALWAYS protect your settings.py file from unauthorized access!
- names used in settings are relative to `os.path`, i.e., the PYTHONPATH environment variable

Basic settings

- `DATABASES`: ENGINE, HOST, NAME, etc.: define databases backend settings
- `INSTALLED_APPS`: Django apps that are used in your project
- `MIDDLEWARE_CLASSES`: middleware classes that process every request
- `ROOT_URLCONF`: the root `url.py` file of your project
- `TEMPLATE_DIRS`: absolute locations of the template source files, in search order. Always use unix-style forward slashes, even on Windows

There are MANY more, but luckily Django provides defaults.

Django applications

Django applications

Creating a Django application:

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

or

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

- Django apps can live anywhere in the filesystem, but must be reachable from `os.path`.
- Contents of `myapp`:
  - `__init__.py`: An empty file that tells Python that this directory should be considered a Python package.
  - `models.py`: This is where you define models for the application.
  - `tests.py`: Tests that will be run when `manage.py test` is invoked.
  - `views.py`: View functions used by the application.

Django's views

- In Django, a view is a function/method that takes at least one argument: the request
- A view should return a `HttpResponse` object
- View functions are usually defined in the application's `views.py`

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

- Rather than hooking a view directly into the project's `urls.py`, an application should define its own `urls.py`

```
1  from django.conf.urls.defaults import *
2  urlpatterns = patterns('mysite.myapp.views',
3      (r'^hello_world/$', 'hello_world_view'),
4  )
```

Project urls vs. app urls

The urls defined in `myapp/urls.py` can be included in the project's `urls.py` as follows:

```
1  from django.conf.urls.defaults import *
2
3  # Uncomment the next two lines to enable the admin:
4  from django.contrib import admin
5  admin.autodiscover()
6
7  urlpatterns = patterns('',
8      # Note how we include the urls defined for myapp:
9      (r'^mysite/', include('mysite.myapp.urls')),
10
11     # Uncomment the next line to enable the admin:
12     (r'^admin/', include(admin.site.urls)),
13 )
```

This allows to easily "plug" applications into various sites. Note the difference between the two includes (quotation marks)!

Activating Django applications

To activate a Django application in a Django project, just add it to the `INSTALLED_APPS` setting:

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

NOTE: here, the application was created inside the project directory.

Django apps are "pluggable": You can use an app in multiple projects, and you can distribute apps, because they don't have to be tied to a given Django project.

Applications bundled with Django

- `django.contrib.auth`: An extensible authentication system
- `django.contrib.admin`: The dynamic administrative interface.
- `django.contrib.comments` A flexible commenting system.
- `django.contrib.sites` A sites framework that allows one Django installation to run multiple websites, each with their own content and applications
- `django.contrib.sitemaps`: Tools for generating Google Sitemaps.
- `django.contrib.csrf.middleware.CsrfMiddleware`: Tools for preventing cross-site request forgery.
- `django.contrib.flatpages`: An app for storing simple “flat” HTML content.
- . . . and more.

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

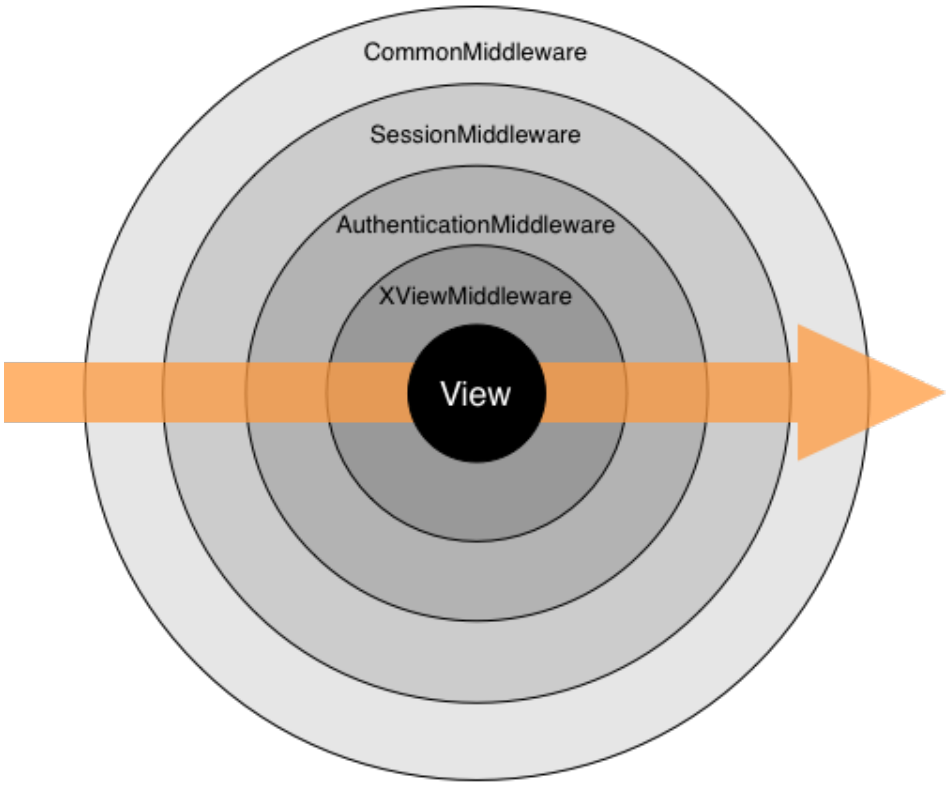
Activating middleware:

```
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

Custom django-admin commands

- Applications can register their own actions with `manage.py`.
- A custom action should be defined as a Python module placed in the `management/commands` directory of an app.
- Each Python module in that directory will be auto-discovered and registered as a command that can be executed as an action when you run `manage.py`

If we place `custom.py` with the following code in the `myapp` directory...

```
1  from django.core.management.base import NoArgsCommand
2  class Command(NoArgsCommand):
3      def handle_noargs(self, **options):
4          print 'Executing custom command...'
5          # command code follows...
```

...then `./manage.py custom` will print:

```
1  Executing custom command...
```

and execute the code defined in the `handle_noargs` method.

References

- http://en.wikipedia.org/wiki/Django_(web_framework)
- Django docs - http://docs.djangoproject.com/en/dev/