Python in the Enterprise
Django Intro

Tomasz Szumlak

WFiIS AGH

23/10/2017, Kraków

Going beyond…

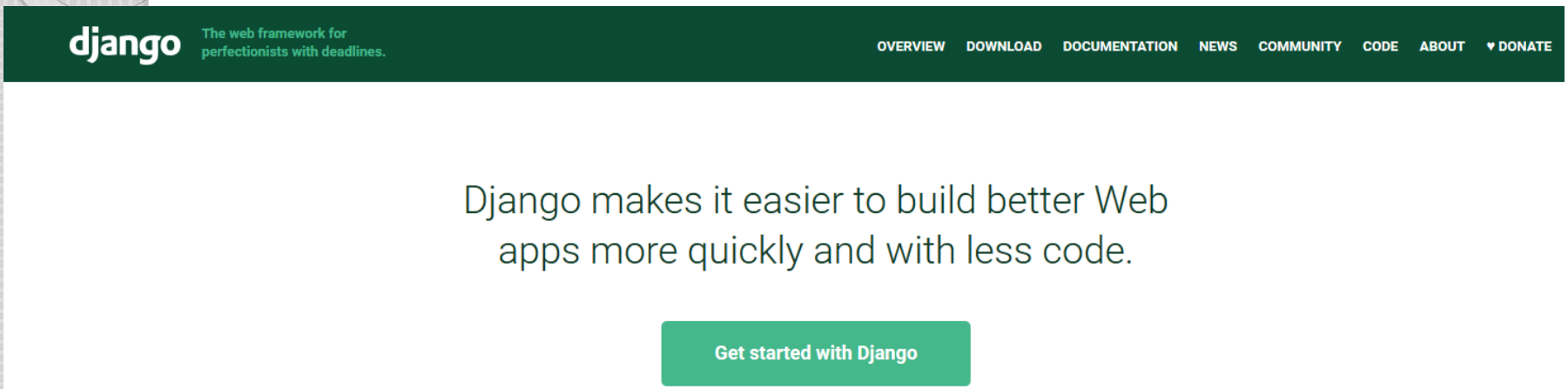
- Django is a **Web framework** – very popular!
- It is not the only one, and cannot do wonders…
- There are **many others**: Flask, TurboGears, web2py…
- Some tasks are easier done with other frameworks, but still Django should be considered one of the most recognised and respected platform
- Now, this is game for serious people… **well past** the Python **standard library**…

Why should I…?

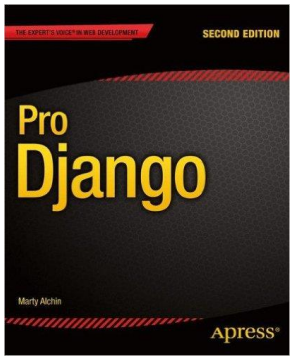
- **Django ORM** (Object-Relational Mapping) is a powerful friend when comes to DB handling, can work with almost all there is on the market…
- A lot of **automatisation** regarding form creation, can have some spare time for living
- **Admin module**, almost ready for deployment with your final application – one of my favourite bits
- Django makes things much, much **easier** (**DRY paradigm**)
- Django is not PHP but has a nice community that can be very helpful
- **Finally – Django is Python**!

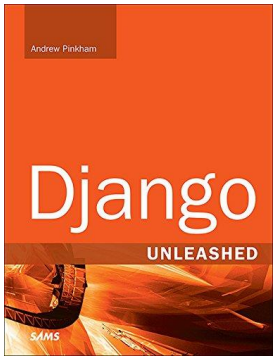
Resources

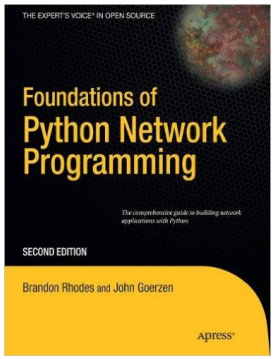
- Django Web page: https://www.djangoproject.com/



- There are some nice books worth looking at, e.g.,







What is a Web framework…?

lansa.com

- All of these presented above you can do **by yourself**, sure! But why don't take an advantage and use vast **experience of others**?
- This is done via Web frameworks – you can use it to create, update and scale-up your apps

However, do not be too lazy…

- Web frameworks are here to help – not to do your job!
- While a lot of tasks **can be hidden** under the hood, you still need to worry about many things:
  - you may want to use **other languages** to implement your business logic (e.g., JavaScript)
  - **low level i-face** to DB is also your burden to carry
  - **application front** (web templating system)
  - Django will help with this but still, you need to think about performing **unit tests**…
- So, the framework can do a lot, but there are still load of work to do!

Before you Django-it-up

- Configuring your workspace is **as important as learning** all tweaks and tricks about Django
- Depending on what is your style and OS you may chose from many different approaches – one of the most fancy is using **virtualenv** workspaces
- Can use on both Win and Linux OS boxes
- Basically you create with this tool completely **isolated environments** for Python development
- Can easily work on many projects at the same time using Python 2.X.X/3.X.X, different libraries etc.
- Just go and have a look at: https://virtualenv.pypa.io/en/stable/

virtualenv hands-on

- Say, we want to use two different versions of Django
- We start with installing **pip** and **virtualenv** (may need root privilages…)

```
[szumlat@xelnaagha22mv dj]$ curl "https://bootstrap.pypa.io/get-pip.py" -o "get-pip.py"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 1558k  100 1558k    0     0  1693k      0 --:--:-- --:--:-- --:--:-- 1693k
[szumlat@xelnaagha22mv dj]$ python get-pip.py
Collecting pip
  Downloading pip-9.0.1-py2.py3-none-any.whl (1.3MB)
    100% |████████████████████████████████| 1.3MB 1.1MB/s
Collecting wheel
  Downloading wheel-0.29.0-py2.py3-none-any.whl (66kB)
    100% |████████████████████████████████| 71kB 5.0MB/s
Installing collected packages: pip, wheel

[root@lhcbgpu1 szumlat]# pip install virtualenv
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |████████████████████████████████| 1.8MB 474kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
```

virtualenv hands-on

- Say, we want to use two different versions of Django

```
[szumlat@xelnaagha22mv devel]$ virtualenv proj1
New python executable in /home/szumlat/dj/devel/proj1/bin/python
Installing setuptools, pip, wheel...done.
[szumlat@xelnaagha22mv devel]$ virtualenv proj2
New python executable in /home/szumlat/dj/devel/proj2/bin/python
Installing setuptools, pip, wheel...done.


[szumlat@xelnaagha22mv devel]$ source proj1/bin/activate
(proj1) [szumlat@xelnaagha22mv devel]$ pip install Django==1.10
Collecting Django==1.10
  Using cached Django-1.10-py2.py3-none-any.whl
Installing collected packages: Django
Successfully installed Django-1.10
(proj1) [szumlat@xelnaagha22mv devel]$ python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(1, 10, 0, u'final', 1)
>>>
(proj1) [szumlat@xelnaagha22mv devel]$ deactivate
[szumlat@xelnaagha22mv devel]$
```

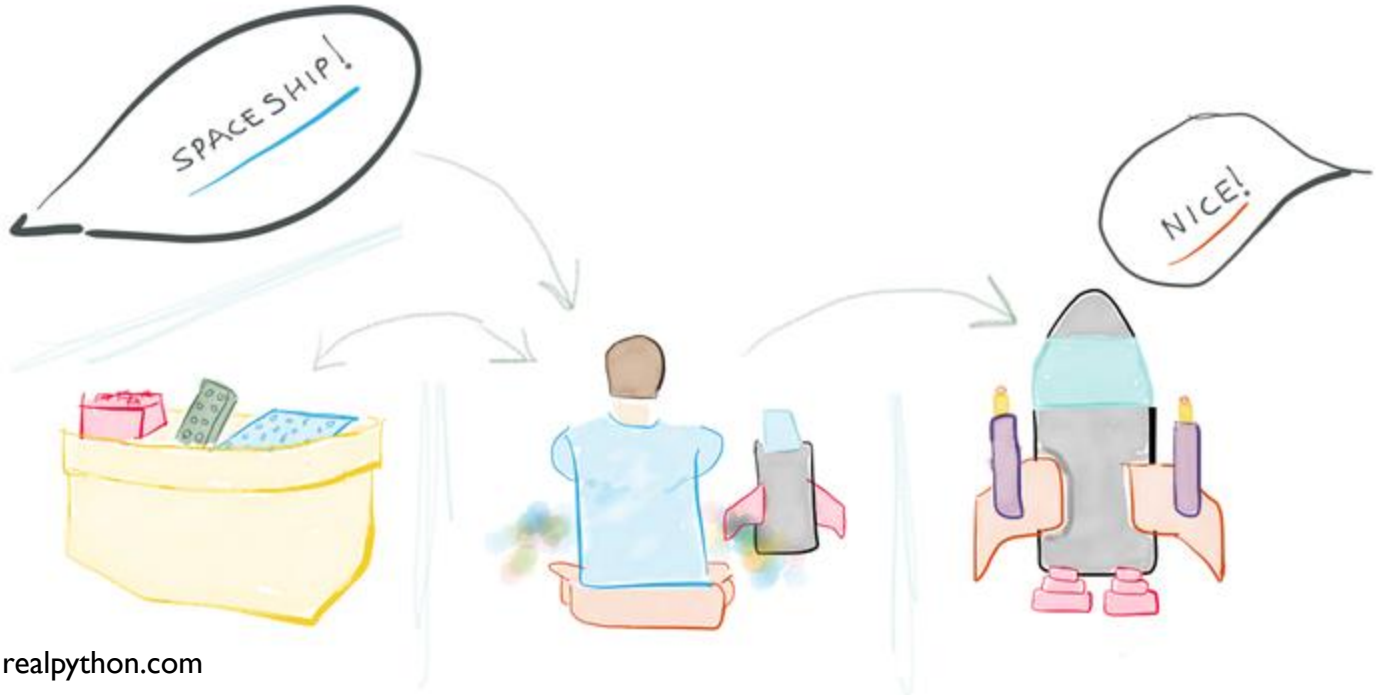
virtualenv hands-on

- Say, we want to use two different versions of Django

```
[szumlat@xelnaagha22mv devel]$ source proj2/bin/activate
(proj2) [szumlat@xelnaagha22mv devel]$ pip install Django==1.10.3
Collecting Django==1.10.3
  Using cached Django-1.10.3-py2.py3-none-any.whl
Installing collected packages: Django
Successfully installed Django-1.10.3
(proj2) [szumlat@xelnaagha22mv devel]$ python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
>>> django.VERSION
(1, 10, 3, u'final', 0)
>>>
(proj2) [szumlat@xelnaagha22mv devel]$ deactivate
[szumlat@xelnaagha22mv devel]$ python
Python 2.7.5 (default, Nov 20 2015, 02:00:19)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import django
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named django
>>>
[szumlat@xelnaagha22mv devel]$
```

Django – Mr. Fancy Pants

- Whole Django rests on various patterns – one of them is **Model-View-Controller** (MVC)
- **Command** pattern can also be seen from time to time



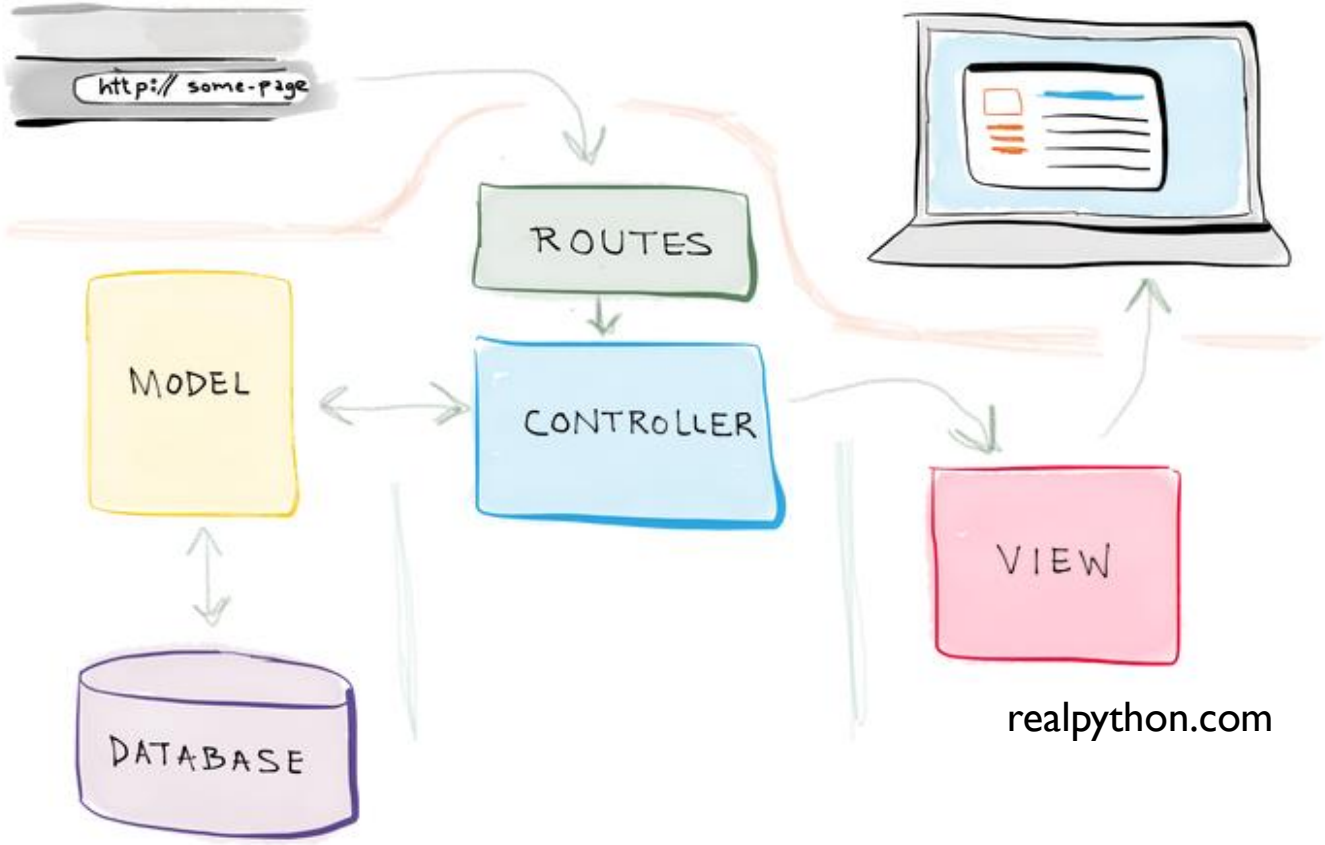
realpython.com

Django – Mr. Fancy Pants

- Saying the same using more Django language
  - A **client interacting** with a web page (front-end)
  - **Controller** module **receives** the client's **request**
  - Using **Models**, the **Controller fetches** the appropriate data **process** them and **format** the results and **sends them out**
  - The **final results** are used by the **View** module to **render** the final web page and present them to the client

Django – Mr. Fancy Pants

- Whole Django rests on various patterns – one of them is **Model-View-Controller** (MVC)



MVC and Django

- The central point of the Django Web application is the **Controller**
  - It handles requests and responses
  - Setting connection to the DB
  - Two main config ingredients: settings file and URL file, first tells it what to set up and load, the second tells what to do with requests
- **Model layer** – basically DB and code that makes queries to it
  - Here models are just Python code that reflects the content of the DB tables
- View – should be thought of as the **user i-face**

First project

- A quick start using Django

```
[szumlat@xelnaagha22mv proj2]$ source bin/activate
(proj2) [szumlat@xelnaagha22mv proj2]$ django-admin startproject myblog
(proj2) [szumlat@xelnaagha22mv proj2]$ ls

myblog/:
  manage.py
  myblog/:
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

- **manage.py** – command line utility to interact with the project (**thin wrapper** around the django-admin tool)
- **myblog/** – folder consisting of the source code of your project
- **__init__.py** – that is clear, this one just states that myblog is a folder…

First project

- A quick start using Django

```
[szumlat@xelnaagha22mv proj2]$ source bin/activate
(proj2) [szumlat@xelnaagha22mv proj2]$ django-admin startproject myblog
(proj2) [szumlat@xelnaagha22mv proj2]$ ls

myblog/:
  manage.py
  myblog/:
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

- **settings.py** – initial default settings for the project
- **urls.py** – the place where your URL patterns are stored, each URL here is mapped to a view
- **wsgi.py** – configuration to run the project as a WSGI application

Default settings and DB

- **settings.py** – created **by default** when a project backbone is created
  - configure to use SQLite
  - collection of the default Django apps that is always a part of each project
- At the beginning we need to create a basic structure of your database, which corresponds to these initial setup

```
(proj2) [szumlat@xelnaagha22mv myblog]$ python manage.py migrate
```

- Remember this is just initialisation, need still some work to actually make it usable!

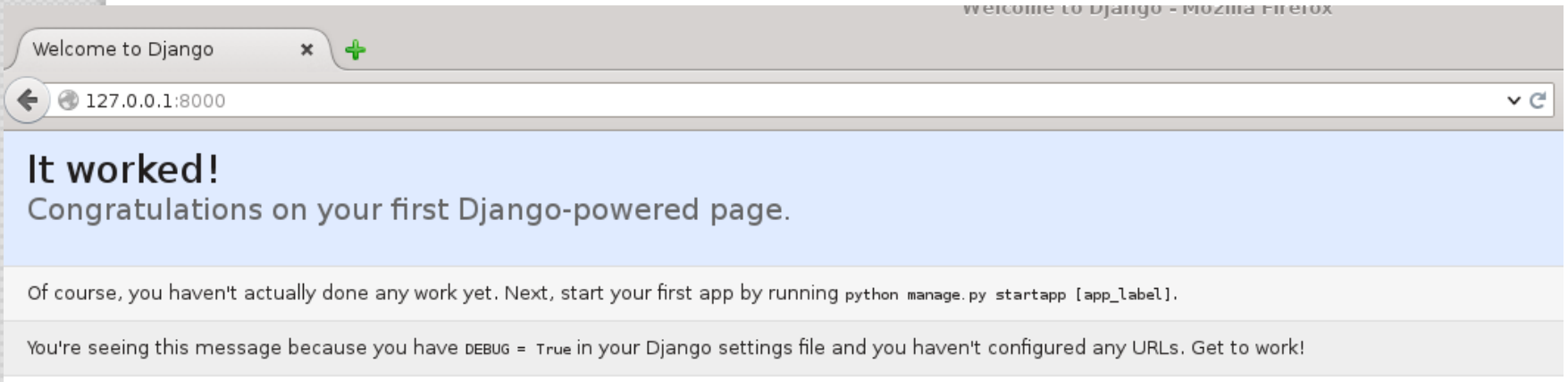
Development server

- This is **one** of the **best features** of Django framework
- A lightweight web server
  - Run quickly you code
  - No need to configure a final production server, which…
    - can be **taxing and frustrating**
  - Also, often you are not a super-user of the machine on which the coding is running
- It can keep track of changes in your code
- No need to **reload** when make modifications
- May miss new files, though! So, each time you add a file or folder restart the Django server manually

Development server

```
(proj2) [szumlat@xelnaagha22mv myblog]$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
December 01, 2016 - 15:58:57
Django version 1.10.3, using settings 'myblog.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
[01/Dec/2016 15:59:04] "GET / HTTP/1.1" 200 1767
Not Found: /favicon.ico
[01/Dec/2016 15:59:04] "GET /favicon.ico HTTP/1.1" 404 1936
```



Configuration

- Let's have a look at the settings.py file

```
"""
Django settings for blog project.

Generated by 'django-admin startproject' using Django 1.10.6.

For more information on this file, see
https://docs.djangoproject.com/en/1.10/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/1.10/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))


# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []
```

In dev mode this is set to true

In dev mode this is always empty

Configuration

- Let's have a look at the settings.py file

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}

# Internationalization
# https://docs.djangoproject.com/en/1.10/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'
```

Here you will add your stuff, says which apps are active for this site

Settings for all DBs to be used in the project – there always must be a default entry

Applications

- Two essential words when playing with Django are
  - Projects
  - Applications
- Projects can be understood as a concrete Django installation with some custom settings
- Application is a collection of models, views, templates and URLs. App interacts with the framework and provides a specific functionality
  - Application **can be shared** in different projects
- So, a Django project can be viewed as a web site, which uses a **number of apps to provide content**

Application – go create!

- We use the manager again:

python manage.py startapp diary_engine

From here…

```
drwxr-xr-x. 2 szumlat z5  4096 Mar 27 01:05 blog
-rw-r--r--. 1 szumlat z5 36864 Mar 27 01:04 db.sqlite3
-rwxr-xr-x. 1 szumlat z5   802 Mar 27 01:04 manage.py
(my_portal) [szumlat@lhcb1 blog]$ python manage.py startapp diary_engine
(my_portal) [szumlat@lhcb1 blog]$ ls -l
drwxr-xr-x. 2 szumlat z5  4096 Mar 27 01:05 blog
-rw-r--r--. 1 szumlat z5 36864 Mar 27 01:04 db.sqlite3
drwxr-xr-x. 3 szumlat z5  4096 Mar 27 12:13 diary_engine
-rwxr-xr-x. 1 szumlat z5   802 Mar 27 01:04 manage.py
(my_portal) [szumlat@lhcb1 blog]$ cd diary_engine/
-rw-r--r--. 1 szumlat z5   63 Mar 27 12:13 admin.py
-rw-r--r--. 1 szumlat z5  139 Mar 27 12:13 apps.py
-rw-r--r--. 1 szumlat z5    0 Mar 27 12:13 __init__.py
drwxr-xr-x. 2 szumlat z5 4096 Mar 27 12:13 migrations
-rw-r--r--. 1 szumlat z5   98 Mar 27 12:13 models.py
-rw-r--r--. 1 szumlat z5   60 Mar 27 12:13 tests.py
-rw-r--r--. 1 szumlat z5   63 Mar 27 12:13 views.py
```

Backbone

Application – go create!

- The structure of the package
  - **admin.py** – for registering models to include into Django administration site
  - **migrations** – data base migrations, allows to track any changes in models and synchronisation with the data base
  - **models.py** – data models for a concrete application, all Django apps need to have this file (however, it can be empty…)
  - **tests.py** – clear…
  - **views.py** – the business logic implementation, each view receives a HTTP request, process it and return the response

Data base schema

- This is a vital step – data models for our diary. A model is formally a class where you define a data base field
- Django will automatically create a table for every model defined in **models.py**
- An example model implementation is below
- Our entry model class inherits from Django Model class
- It is not only a mere convenience, it allows to use a very rich API to work with the data base
- Imagine you implement this all from scratch!

Data base schema

```
from __future__ import unicode_literals
from django.db import models
# Create your models here.

from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

class diary_entry( models.Model ):

  STATUS_CHOICES = (
    ('draft', 'Draft'),
    ('published', 'Published'),
  )
  title = models.CharField( max_length = 250 )
  slug = models.SlugField( max_length = 250, unique_for_date = 'published')
  author = models.ForeignKey( User, related_name = 'diary_entry' )
  body = models.TextField()
  publish = models.DateTimeField( default = timezone.now )
  created = models.DateTimeField( auto_now_add = True )
  updated = models.DateTimeField( auto_now = True )
  status = models.CharField( max_length = 10, choices = STATUS_CHOICES,
                             default = 'draft' )

class Meta:
  ordering = ( '-publish' )

def __str__( self ):
  return ( self.title )
```

Data base schema

- What can we find inside…
  - **title** – this will hold a post title (**VARCHAR** in SQL)
  - **slug** – this is for URLs, a label that help in SEO (Search Engine Optimisation). We use it to build URLs for posts using the date and slug
  - **author** – it is **ForeignKey** field, defines a many-2-one relationship: each post is written by a user and each user can produce many, many posts… Django will create a key in its DB using User model of the Django authentication system
  - **body** – body of the entry, corresponds to a TEXT column in SQL
  - **publish** – this **datetime** indicates when the post was poublished we use **timezone** sensitive **datetime.now**

Data base schema

- As you can see such model needs to take into account a lot of stuff!
- The full description can be found here:

  https://docs.djangoproject.com/en/1.10/ref/models/fields

- The class Meta contains metadata. We are telling Django how to cope with sorting – we want to use the publish field in descending order – this is the default behaviour when browsing the data base
- The last ingredient **__str__( )** – this one is to provide the default human-readable representation of the object

Activation…

- Add it to the list of the installed applications

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'diary_engine',
]
```

- Django framework now know that the application is active for this project and will be able to interact with its models