

Django - Framework



What is Django?

Web Development Framework - based on Python Programming

Why Django?

With Django, you can take Web applications from concept to launch in a matter of hours.

Django takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

- Fast
- Fully Loaded
- Secure
- Scalable
- Versatile - used for multiple purposes

Easy - But not so easy, But Large Community

Infrastructure

```
root@531d68e38300:/workshop/workshop_one/django_workshop# pip3 install virtualenv
Collecting virtualenv
  Downloading virtualenv-15.1.0-py2.py3-none-any.whl (1.8MB)
    100% |#####| 1.8MB 127kB/s
Installing collected packages: virtualenv
Successfully installed virtualenv-15.1.0
```

```
root@531d68e38300:/workshop/workshop_one# virtualenv django_workshop
Using base prefix '/usr'
New python executable in /workshop/workshop_one/django_workshop/bin/python3
Also creating executable in /workshop/workshop_one/django_workshop/bin/python
Installing setuptools, pip, wheel...done. _
```

```
root@531d68e38300:/workshop/workshop_one# source django_workshop/bin/activate
(django_workshop) root@531d68e38300:/workshop/workshop_one#
```

```
(django_workshop) root@531d68e38300:/workshop/workshop_one# python --version
Python 3.5.2
```

Infrastructure (Cont ...)

```
(django_workshop) root@531d68e38300:/workshop/workshop_one# python -c "import django; print(django.get_version())"
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ImportError: No module named 'django'
(django_workshop) root@531d68e38300:/workshop/workshop_one# which pip3
/workshop/workshop_one/django_workshop/bin/pip3
(django_workshop) root@531d68e38300:/workshop/workshop_one# pip3 install django
Collecting django
  Downloading Django-1.11.2-py2.py3-none-any.whl (6.9MB)
    100% |#####| 7.0MB 79kB/s
Collecting pytz (from django)
  Using cached pytz-2017.2-py2.py3-none-any.whl
Installing collected packages: pytz, django
Successfully installed django-1.11.2 pytz-2017.2
(django_workshop) root@531d68e38300:/workshop/workshop_one# python -c "import django; print(django.get_version())"
1.11.2
(django_workshop) root@531d68e38300:/workshop/workshop_one#
```

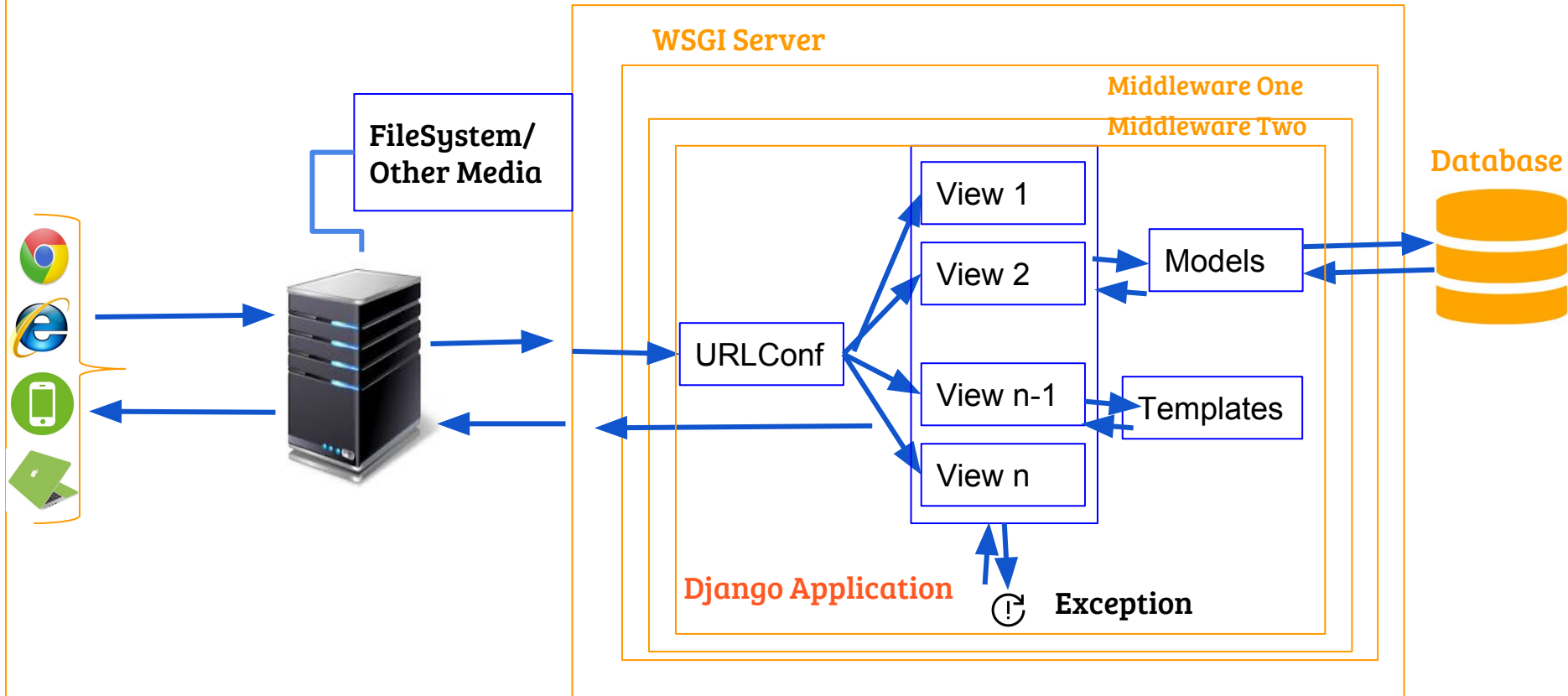
Starting Project

```
$ django-admin startproject django_workshop
$ tree django_workshop
django_workshop
|-- django_workshop
|   |-- __init__.py
|   |-- settings.py
|   |-- urls.py
|   '-- wsgi.py
'-- manage.py

1 directory, 5 files
```

- **`__init__.py`** : That's blank Python Script. Says it's Python Package. Every module is available for **import**
- **`settings.py`** - Store your Django project's settings
- **`urls.py`** - a Python script to store URL patterns for your project
- **`wsgi.py`** - Important to deploy project in production environment.

Request Processing in Django Application



Starting Project (Cont ...)

```
$ cd django_workshop/  
$ pwd  
/workshop/workshop_one/django_workshop/django_workshop  
$ python manage.py runserver  
Performing system checks...  
  
System check identified no issues (0 silenced).
```

You have 13 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.

```
June 24, 2017 - 07:03:29  
Django version 1.11.2, using settings 'django_workshop.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

```
$ sqlite3 db.sqlite3  
SQLite version 3.11.0 2016-02-15 17:29:24  
Enter ".help" for usage hints.  
sqlite> .schema  
CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL, "applied" datetime NOT NULL);  
sqlite>
```

You have no database tables (just one here)

Starting Project (Cont ...)

```
$ python manage.py migrate
```

```
Operations to perform:
```

```
Apply all migrations: admin, auth, contenttypes, sessions
```

```
Running migrations:
```

```
Applying contenttypes.0001_initial... OK
```

```
Applying auth.0001_initial... OK
```

```
Applying admin.0001_initial... OK
```

```
Applying admin.0002_logentry_remove_auto_add... OK
```

```
Applying contenttypes.0002_remove_content_type_name... OK
```

```
Applying auth.0002_alter_permission_name_max_length... OK
```

```
Applying auth.0003_alter_user_email_max_length... OK
```

```
Applying auth.0004_alter_user_username_opts... OK
```

```
Applying auth.0005_alter_user_last_login_null... OK
```

```
Applying auth.0006_require_contenttypes_0002... OK
```

```
Applying auth.0007_alter_validators_add_error_messages... OK
```

```
Applying auth.0008_alter_user_username_max_length... OK
```

```
Applying sessions.0001_initial... OK
```

```
$ █
```


Starting Project (Cont ...)

```
sqlite> .schema
CREATE TABLE "django_migrations" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "app" varchar(255) NOT NULL, "name" varchar(255) NOT NULL, "applied" datetime NOT NULL);
CREATE TABLE "auth_group" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "name" varchar(80) NOT NULL);
CREATE TABLE "auth_group_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "group_id" integer NOT NULL REFERENCES "auth_group" ("id"), "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id"));
CREATE TABLE "auth_user_groups" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id"), "group_id" integer NOT NULL REFERENCES "auth_group" ("id"));
CREATE TABLE "auth_user_user_permissions" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "user_id" integer NOT NULL REFERENCES "auth_user" ("id"), "permission_id" integer NOT NULL REFERENCES "auth_permission" ("id"));
CREATE UNIQUE INDEX "auth_group_permissions_group_id_permission_id_0cd325b0_uniq" ON "auth_group_permissions" ("group_id", "permission_id");
CREATE INDEX "auth_group_permissions_group_id_b120cbf9" ON "auth_group_permissions" ("group_id");
CREATE INDEX "auth_group_permissions_permission_id_84c5c92e" ON "auth_group_permissions" ("permission_id");
CREATE UNIQUE INDEX "auth_user_groups_user_id_group_id_94350c0c_uniq" ON "auth_user_groups" ("user_id", "group_id");
CREATE INDEX "auth_user_groups_user_id_6a12ed8b" ON "auth_user_groups" ("user_id");
CREATE INDEX "auth_user_groups_group_id_97559544" ON "auth_user_groups" ("group_id");
CREATE UNIQUE INDEX "auth_user_user_permissions_user_id_permission_id_14a6b632_uniq" ON "auth_user_user_permissions" ("user_id", "permission_id");
```

You have Many Database Tables Now!

Starting Project (Cont ...)

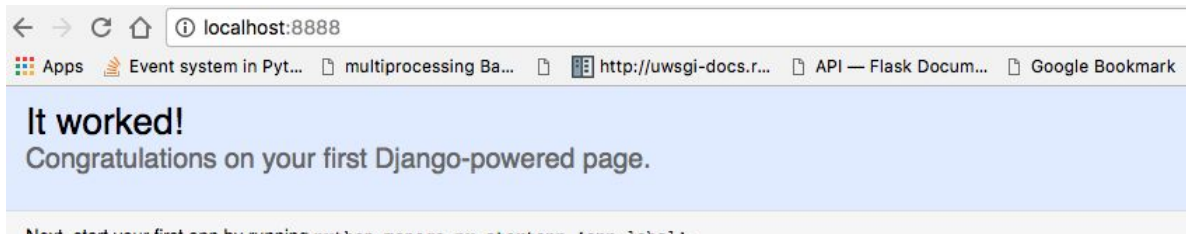
```
$ python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
June 24, 2017 - 11:50:40
Django version 1.11.2, using settings 'django_workshop.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Let us change the port to 8888

```
$ python manage.py runserver 0:8888
Performing system checks...

System check identified no issues (0 silenced).
June 24, 2017 - 11:52:58
Django version 1.11.2, using settings 'django_workshop.settings'
Starting development server at http://0:8888/
Quit the server with CONTROL-C.
[24/Jun/2017 11:53:11] "GET / HTTP/1.1" 200 1716
```



Starting Project (Cont ...)

You have additional Files too

```
$ tree django_workshop/
django_workshop/
|-- __init__.py
|-- __pycache__
|   |-- __init__.cpython-35.pyc
|   |-- settings.cpython-35.pyc
|   |-- urls.cpython-35.pyc
|   `-- wsgi.cpython-35.pyc
|-- settings.py
|-- urls.py
`-- wsgi.py

1 directory, 8 files
$
```

Starting Application

```
$ python manage.py startapp dravate
```

```
$ tree
.
|-- db.sqlite3
|-- django_workshop
|   |-- __init__.py
|   |-- __pycache__
|   |   |-- __init__.cpython-35.pyc
|   |   |-- settings.cpython-35.pyc
|   |   |-- urls.cpython-35.pyc
|   |   `-- wsgi.cpython-35.pyc
|   |-- settings.py
|   |-- urls.py
|   `-- wsgi.py
|-- dravate
|   |-- __init__.py
|   |-- admin.py
|   |-- apps.py
|   |-- migrations
|   |   |-- __init__.py
|   |-- models.py
|   |-- tests.py
|   `-- views.py
`-- manage.py
```

4 directories, 17 files

Starting Application (Cont ...)

- **__init__.py** - serving the exact same purpose as discussed previously
- **models.py**- a place to store your application's data models
- **tests.py**- where you can store a series of functions to test your application
- **views.py** - where you can store a series of functions that take a client's requests and return responses.
- **admin.py** - where you can register your models so that you can benefit from some Django machinery which creates an admin interface for you

Starting Application (Cont ...)

[django_workshop/settings.py](#)

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


Starting Application (Cont ...)

dravate/views.py

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def index(request):
    return HttpResponse("Dravate says hey there world!")
```

dravate/urls.py

```
from django.conf.urls import url
from dravate import views

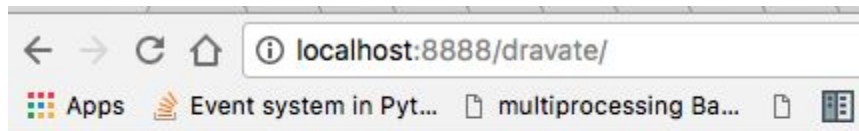
urlpatterns = [url(r'^$', views.index, name='index'),]
```

Starting Application (Cont ...)

[django_workshop/urls.py](#)

```
from django.conf.urls import url, include
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^dravate/', include('dravate.urls')),
]
```



Dravate says hey there world!

Starting Application - Summary of Actions

1. `$ python manage.py startapp <appname>` - Helps create new application
2. Tell your Django project about the new application - Edit `INSTALLED_APPS` tuple in your project's `settings.py` file.
3. Add `urls.py` file in your application directory
4. Modify your project "url.py" - add mapping to your new application (via include)
5. In your application's `view.py`, create the required views ensuring that they return a `HttpResponse` object.

Task - 1

Create “`http://localhost:8888/dravate/about`” view

- Add about method in `dravate/views.py` module
- Update `dravate/urls.py` to support `dravate/about` mapping
- Do you need that restart in your development server?

Templates and Static Media

```
$ tree templates/  
templates/  
|-- dravate  
    |-- index.html  
  
1 directory, 1 file
```

```
$ more templates/dravate/index.html  
<!DOCTYPE html>  
<html>  
  
    <head>  
        <title>Dravate</title>  
    </head>  
  
    <body>  
        <h1>Dravate says...</h1>  
        hello world! <strong>{{ boldmessage }}</strong><br />  
        <a href="/dravate/about/">About</a><br />  
    </body>  
  
</html>
```

Templates and Static Media (Cont ...)

dravate/views.py

```
from django.shortcuts import render

# Create your views here.
from django.http import HttpResponse

def index(request):
    #return HttpResponse("Dravate says hey there world!")
    context_dict = {'boldmessage': "I am bold font from the context"}
    return render(request, 'dravate/index.html', context_dict)
```



Templates and Static Media (Cont ...)

[django_workshop/settings.py](#)

```
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, "static"),  
]
```

[templates/dravate/index.html](#)

```
<!DOCTYPE html>
```

```
{% load static %} <!-- New line -->
```

```
<html>
```

```
    <head>
```

```
        <title>Dravate</title>
```

```
    </head>
```

```
    <body>
```

```
        <h1>Dravate says...</h1>
```

```
        hello world! <strong>{{ boldmessage }}</strong><br />
```

```
        <a href="/dravate/about/">About</a><br />
```

```
         <!-- New line -->
```

```
    </body>
```

```
</html>
```

```
$ tree static/  
static/  
  -- images  
    -- django_demo_name.jpeg  
  
1 directory, 1 file
```

Templates and Static Media (Cont ...)



Templates - Summary of Actions

- Develop template within the `templates` directory
 - Specify templates directory in `settings.py` file.
 - Modify `views.py` methods to support your templates
 - Construct a dictionary object which you can pass to the template engine as part of the template's *context*.
 - Make use of the `render()` helper function to generate the rendered response.
-
- Place static media file in project's `static` directory. The directory details saved in project's `STATICFILES_DIRS` tuple within `settings.py`.
 - Add a reference to the static media file to a template.
 - Remember to use the `{% load staticfiles %}` and `{% static "filename" %}` commands within the template to access the static files.

Task 2

Work on - /dravate/about page - add picture to it.

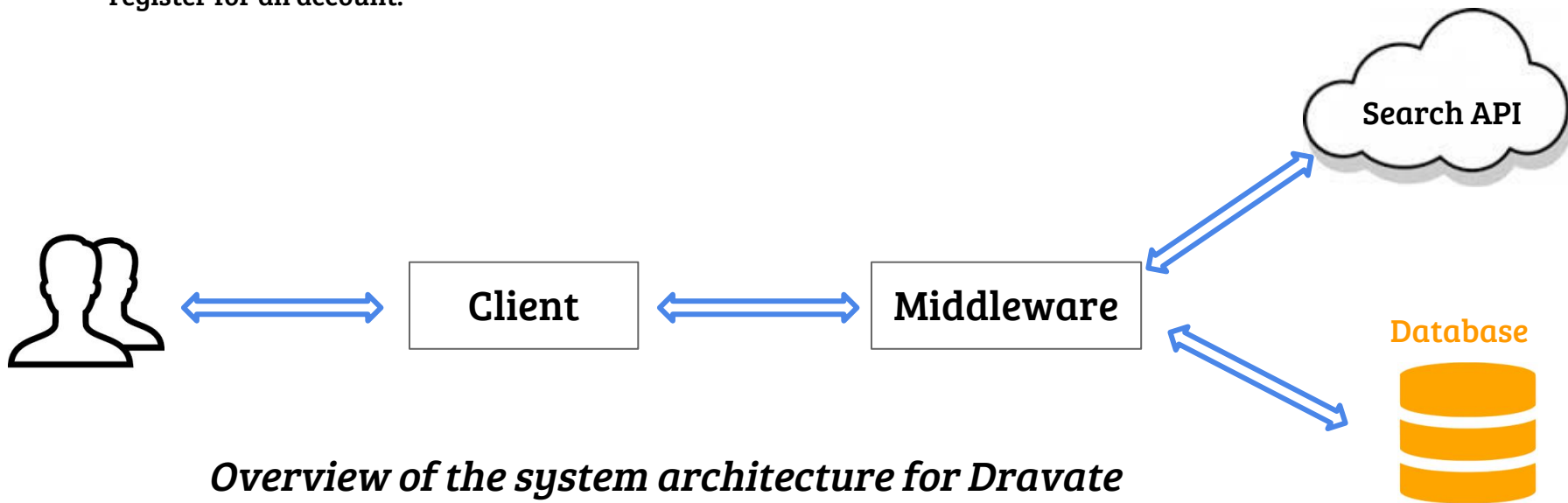
Define Application - Dravate

Let us create a website called *Dravate* that lets users browse through user-defined categories to access various web pages.

- For the *main page* of the site, We would like visitors to be able to see:
 - the 5 most viewed pages;
 - the five most categories; and
 - some way for visitors to browse or search through categories.
- When a user views a *category page*, they would like it to display:
 - the category name, the number of visits, the number of likes;
 - along with the list of associated pages in that category (showing the page's title and linking to its url); and.
 - some search functionality (via Bing's Search API) to find other pages that can be linked to this category.

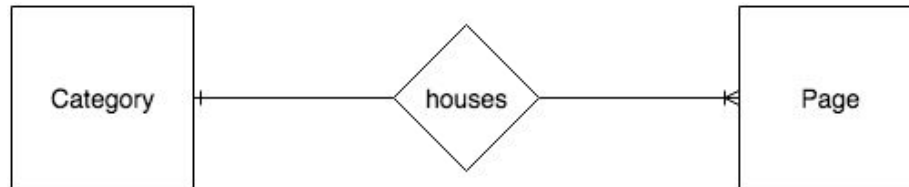
Define Application - Dravate (Cont ...)

- For a particular category, the client would like the name of the category to be recorded, the number of times each category page has been visited, and how many users have clicked a “like” button
- Each category should be accessible via a readable URL - for example, `/dravate/books-about-django/`.
- Only registered users will be able to search and add pages to categories. And so, visitors to the site should be able to register for an account.



Models and Databases

- Almost all hassle is taken care by Django's *object relational mapping (ORM)*
 - Django encapsulates databases tables through models
 - model is a Python object that describes your data model/table
 - That helps you manipulate the Python object than playing with SQL Queries
-
- Dravate is a essentially a *web page directory* - a site containing links to other websites.
 - There are a number of different *webpage categories*, and each category houses a number of links. Entity Relationship Diagram below.
 - A category has a name, number of visits, and number of likes.
 - A page refers to a category, has a title, URL and a number of views.



Models and Databases (Cont ...)

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

[dravate/models.py](#)

[django_workshop/settings.py](#)

Models and Databases (Cont ...)

`dravate/models.py`

```
from django.db import models

class Category(models.Model):
    name = models.CharField(max_length=128, unique=True)

    def __str__(self):
        return self.name

class Page(models.Model):
    category = models.ForeignKey(Category)
    title = models.CharField(max_length=128)
    url = models.URLField()
    views = models.IntegerField(default=0)

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

Django creates an ID field for you automatically in each table relating to a model

Create Tables, Create Superuser

```
$ python manage.py makemigrations
Migrations for 'dravate':
  dravate/migrations/0001_initial.py
    - Create model Category
    - Create model Page
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, dravate, sessions
Running migrations:
  Applying dravate.0001_initial... OK
$
```

```
$ python manage.py createsuperuser
Username (leave blank to use 'root'): admin@dravate.com
Email address: info@dravate.com
Password:
Password (again):
Superuser created successfully.
$
```

localhost:8888/admin/

WELCOME, **ADMIN@DRAVATE.COM**. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

My actions

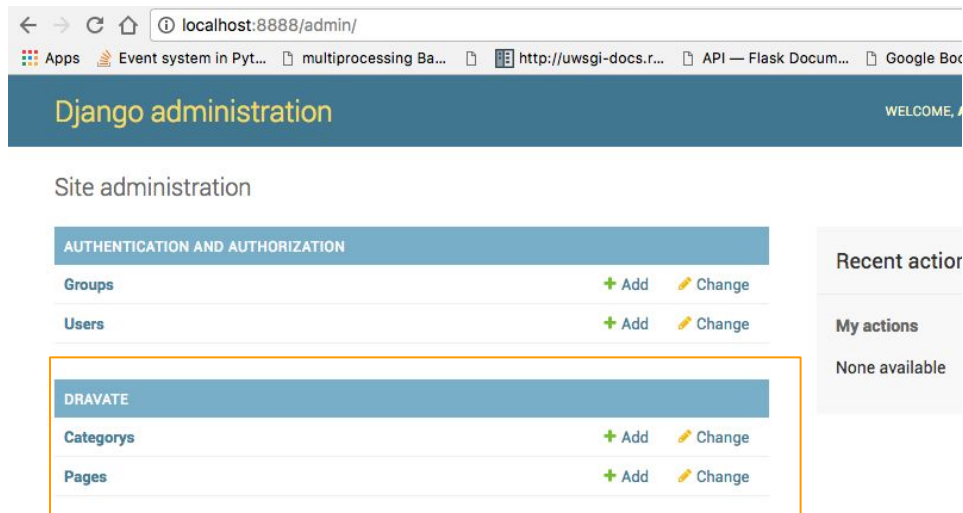
None available

Admin Interface - Manage Models

dravate/admin.py

```
from django.contrib import admin
from dravate.models import Category, Page

admin.site.register(Category)
admin.site.register(Page)
```



The screenshot shows the Django administration interface in a web browser. The address bar indicates the URL is `localhost:8888/admin/`. The page title is "Django administration" with a "WELCOME," message on the right. Below the title, it says "Site administration".

The interface displays a list of model groups. The first group is "AUTHENTICATION AND AUTHORIZATION" with two items: "Groups" and "Users", each with a green plus icon for "Add" and a yellow pencil icon for "Change".

The second group, "DRAVATE", is highlighted with an orange border and contains two items: "Categorys" and "Pages", each with a green plus icon for "Add" and a yellow pencil icon for "Change".

On the right side of the interface, there are two sections: "Recent actions" and "My actions", both showing "None available".

Django Models and Shell Access

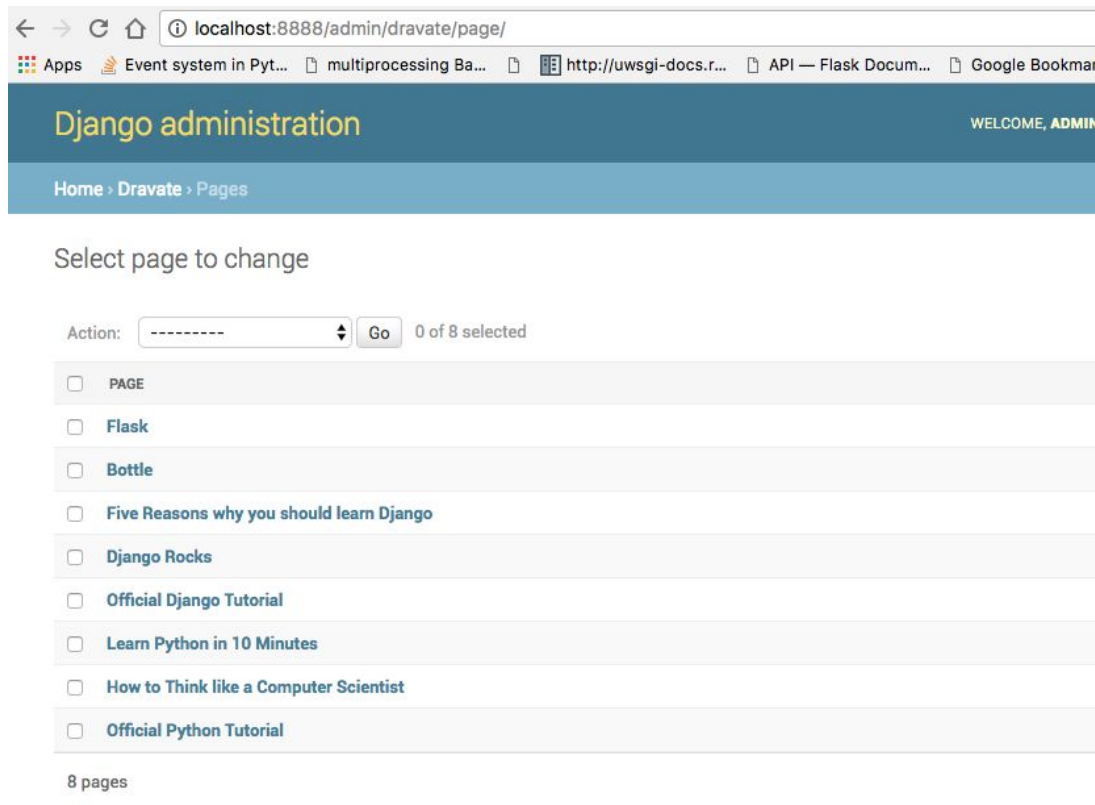
```
$ python manage.py shell
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
(InteractiveConsole)
>>>
>>> from dravate.models import Category
>>> print (Category.objects.all())
<QuerySet [<Category: Python>, <Category: Django>, <Category: Other Frameworks>]>
>>> █
```

You can develop Python Script to upload data or manage other activity

```
$ python populate_dravate.py
Starting Dravate population script...
- Python - Official Python Tutorial
- Python - How to Think like a Computer Scientist
- Python - Learn Python in 10 Minutes
- Django - Official Django Tutorial
- Django - Django Rocks
- Django - Five Reasons why you should learn Django
- Other Frameworks - Bottle
- Other Frameworks - Flask
$
```

[populate_dravate.py](#)

Admin Interface - Data Uploaded



The screenshot shows a web browser window with the address bar at `localhost:8888/admin/dravate/page/`. The browser's tab bar includes "Apps", "Event system in Pyt...", "multiprocessing Ba...", "http://uwsgi-docs.r...", "API — Flask Docum...", and "Google Bookmar". The Django administration interface has a dark blue header with "Django administration" in yellow and "WELCOME, ADMIN" on the right. Below the header is a light blue breadcrumb trail: "Home > Dravate > Pages". The main content area is titled "Select page to change". It features an "Action:" dropdown menu with a "Go" button and a status "0 of 8 selected". A list of 8 pages follows, each with a checkbox and a title: "PAGE", "Flask", "Bottle", "Five Reasons why you should learn Django", "Django Rocks", "Official Django Tutorial", "Learn Python in 10 Minutes", and "Official Python Tutorial". At the bottom, it says "8 pages".

localhost:8888/admin/dravate/page/

Apps Event system in Pyt... multiprocessing Ba... http://uwsgi-docs.r... API — Flask Docum... Google Bookmar

Django administration WELCOME, ADMIN

Home > Dravate > Pages

Select page to change

Action: ----- Go 0 of 8 selected

- ☐ PAGE
- ☐ Flask
- ☐ Bottle
- ☐ Five Reasons why you should learn Django
- ☐ Django Rocks
- ☐ Official Django Tutorial
- ☐ Learn Python in 10 Minutes
- ☐ How to Think like a Computer Scientist
- ☐ Official Python Tutorial

8 pages

Database Models - Summary of Actions

- First, create your new model(s) in your Django application's `models.py` file.
- Update `admin.py` to include and register your new model(s).
- Then perform the migration `$ python manage.py makemigrations`
- Apply the changes `$ python manage.py migrate`. This will create the necessary infrastructure within the database for your new model(s).
- Create/Edit your population script for your new model(s).

Task-3

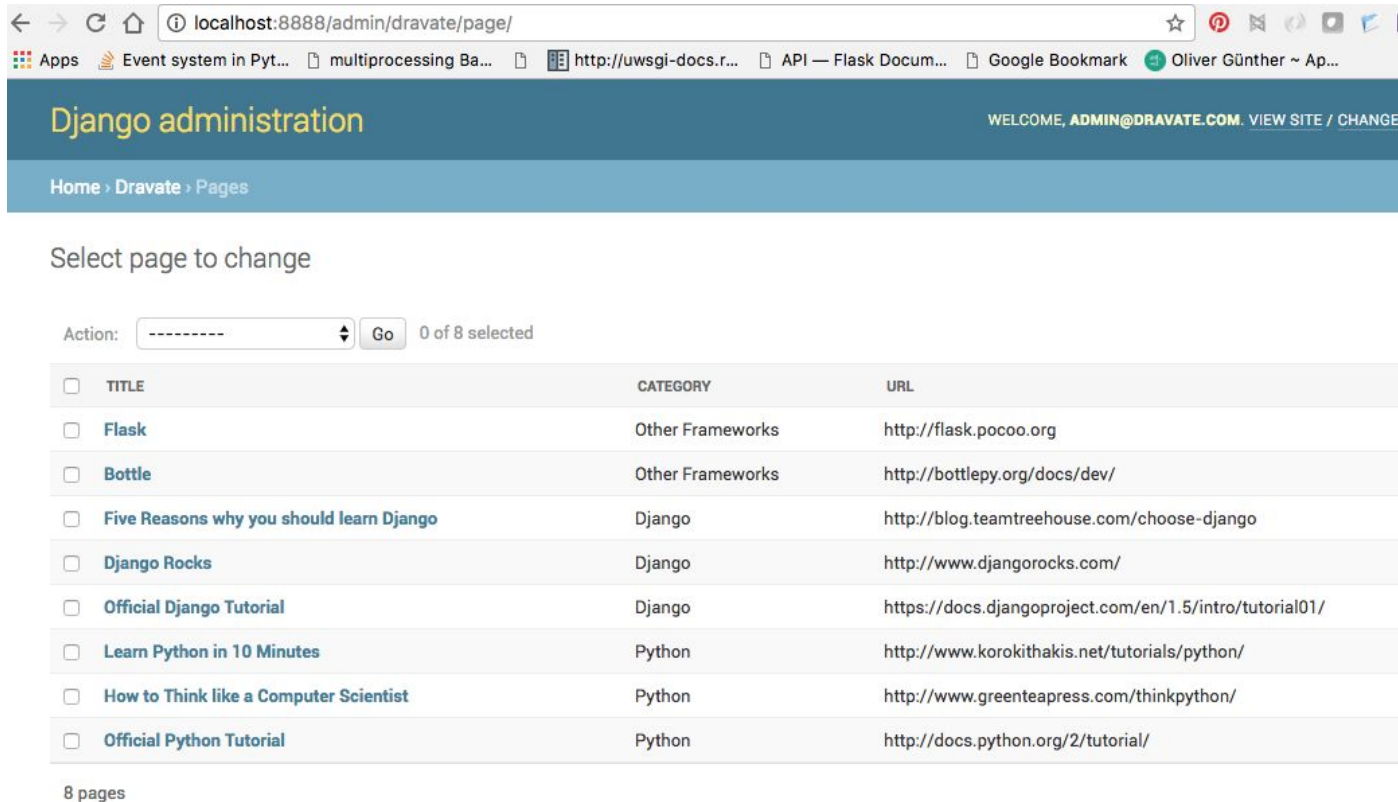
- Update the Category model to include the additional attributes, `views` and `likes` where the default value is zero.
- Make the migrations for your app/model, then migrate your database
- Update your population script so that the Python category has 128 views and 64 likes, the Django category has 64 views and 32 likes, and the Other Frameworks category has 32 views and 16 likes.

Refer - <https://docs.djangoproject.com/en/1.11/intro/tutorial07/>

Task-3 (Help)

- Update - Category model, add fields, view and likes as IntegerFields.
- Modify the add_cat function in the populate_dravate.py script, to take the views and likes.
- In admin interface, edit dravate/admin.py and create a PageAdmin class that inherits from admin.ModelAdmin.
- Within your new PageAdmin class, add list_display = ('title', 'category', 'url').
- Finally, register the PageAdmin class with Django's admin interface. You should modify the line admin.site.register(Page). Change it to admin.site.register(Page, PageAdmin) in dravate admin.py file.

Task - 3 (Expected Look)



The screenshot shows the Django administration interface in a web browser. The address bar indicates the URL is `localhost:8888/admin/dravate/page/`. The page title is "Django administration" and the user is logged in as "ADMIN@DRAVATE.COM". The breadcrumb trail is "Home > Dravate > Pages".

Below the breadcrumb trail, there is a section titled "Select page to change". It includes an "Action:" dropdown menu (currently showing "-----"), a "Go" button, and a status "0 of 8 selected".

The main content area displays a table of pages. The table has three columns: "TITLE", "CATEGORY", and "URL". There are 8 rows of data, each with a checkbox in the "TITLE" column.

<input type="checkbox"/> TITLE	CATEGORY	URL
<input type="checkbox"/> Flask	Other Frameworks	http://flask.pocoo.org
<input type="checkbox"/> Bottle	Other Frameworks	http://bottlepy.org/docs/dev/
<input type="checkbox"/> Five Reasons why you should learn Django	Django	http://blog.teamtreehouse.com/choose-django
<input type="checkbox"/> Django Rocks	Django	http://www.djangorocks.com/
<input type="checkbox"/> Official Django Tutorial	Django	https://docs.djangoproject.com/en/1.5/intro/tutorial01/
<input type="checkbox"/> Learn Python in 10 Minutes	Python	http://www.korokithakis.net/tutorials/python/
<input type="checkbox"/> How to Think like a Computer Scientist	Python	http://www.greenteapress.com/thinkpython/
<input type="checkbox"/> Official Python Tutorial	Python	http://docs.python.org/2/tutorial/

At the bottom left, it says "8 pages".

Models, Templates and Views

- Import the models you wish to use into your application's `views.py` file.
- In view, query the model to get the data you want to present.
- Pass the results from your model into the template's context.
- Setup your template to present the data to the user in whatever way you wish.
- URL mapping to your view is also important - do that too

Models, Templates and Views (Cont ...)

```
from django.shortcuts import render
from django.http import HttpResponse
from dravate.models import Category
```

```
def index(request):
    #return HttpResponse("Dravate says hey there world!")

    category_list = Category.objects.order_by('-likes')[:5]
    context_dict = {'categories': category_list}

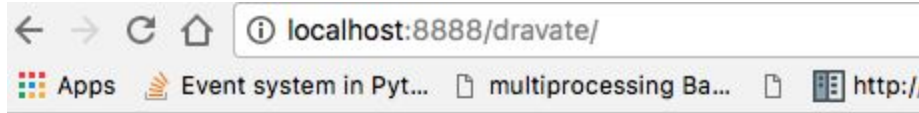
    #context_dict = {'boldmessage': "I am bold font from the context"}
    return render(request, 'dravate/index.html', context_dict)
```

[dravate/views.py](#)

[templates/dravate/index.html](#)

```
{% if categories %}
    <ul>
        {% for category in categories %}
        <li>{{ category.name }}</li>
        {% endfor %}
    </ul>
{% else %}
    <strong>There are no categories present.</strong>
{% endif %}
```


Models, Templates and Views (Cont ...)



Dravate says...

- Python
- Django
- Other Frameworks

[About](#)

Django Django

Our Applications - Show Details Page

show a list of pages that are associated with each category

```
from django.template.defaultfilters import slugify

class Category(models.Model):
    name = models.CharField(max_length=128, unique=True)
    views = models.IntegerField(default=0)
    likes = models.IntegerField(default=0)
    slug = models.SlugField()

    def save(self, *args, **kwargs):
        self.slug = slugify(self.name)
        super(Category, self).save(*args, **kwargs)
```

Our Applications - Show Details Page

```
$ python manage.py makemigrations
You are trying to add a non-nullable field 'slug' to category without a default; we can't do that (the database needs
ing to populate existing rows).
Please select a fix:
  1) Provide a one-off default now (will be set on all existing rows with a null value for this column)
  2) Quit, and let me add a default in models.py
Select an option: 1
Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can do e.g. timezone.now
Type 'exit' to exit this prompt
>>> ''
Migrations for 'dravate':
  dravate/migrations/0003_category_slug.py
    - Add field slug to category
$
```

```
$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, dravate, sessions
Running migrations:
  Applying dravate.0003_category_slug... OK
$
```

```
from django.contrib import admin
from dravate.models import Category, Page

#admin.site.register(Category)
#admin.site.register(Page, PageAdmin)

class PageAdmin( admin.ModelAdmin):
    list_display = ('title', 'category', 'url')

class CategoryAdmin(admin.ModelAdmin):
    prepopulated_fields = {'slug':('name',)}

# Update the registration to include this customised interface
admin.site.register(Category, CategoryAdmin)
admin.site.register(Page, PageAdmin)
~
```

Django administration

Home › Dravate › Categorys

Select category to change

Action: ⬆️⬆️ 0 of 3 selected

- ☐ CATEGORY
- ☐ Other Frameworks
- ☐ Django
- ☐ Python

3 categorys

Application - Category Page

1. Import the Page model into `dravate/views.py`.
2. Create a new view in `dravate/views.py` - called `category` - The `category` view will take an additional parameter, `category_name_url` which will stored the encoded category name.
 - We will need some help functions to encode and decode the `category_name_url`.
3. Create a new template, `templates/dravate/category.html`.
4. Update `urlpatterns` to map the new `category` view to a URL pattern in `dravate/urls.py`.



Add a Category

Please enter the category name.



Dravate says...

- [Python](#)
- [Django](#)
- [Other Frameworks](#)
- [dravate](#)
- [Politics](#)

[About](#)

Django Django

Forms

Django comes with some neat form handling functionality, making it a pretty straightforward process to gather information from users and send it back to your web application

1. display an HTML form with automatically generated *form widgets* (like a text field or date picker);
2. check submitted data against a set of validation rules;
3. redisplay a form in case of validation errors; and
4. convert submitted form data to the relevant Python data types.

Forms (Cont ...) - Workflow

- If you haven't already got one, create a `forms.py` file within your Django application's directory to store form-related classes.
- Create a `ModelForm` class for each model that you wish to represent as a form.
- Customise the forms as you desire.
- Create or update a view to handle the form - including *displaying* the form, *saving* the form data, and *flagging up errors* which may occur when the user enters incorrect data (or no data at all) in the form.
- Create or update a template to display the form.
- Add a `urlpatterns` to map to the new view (if you created a new one).

Forms (Cont ...)

```
from django import forms
from dravate.models import Page, Category

class CategoryForm(forms.ModelForm):
    name = forms.CharField(max_length=128, help_text="Please enter the category name.")
    views = forms.IntegerField(widget=forms.HiddenInput(), initial=0)
    likes = forms.IntegerField(widget=forms.HiddenInput(), initial=0)
    slug = forms.CharField(widget=forms.HiddenInput(), required=False)

    # An inline class to provide additional information on the form.
    class Meta:
        # Provide an association between the ModelForm and a model
        model = Category
        fields = ('name',)
```

```
def add_category(request):
    # A HTTP POST?
    if request.method == 'POST':
        form = CategoryForm(request.POST)

        # Have we been provided with a valid form?
        if form.is_valid():
            # Save the new category to the database.
            form.save(commit=True)

            # Now call the index() view.
            # The user will be shown the homepage.
            return index(request)
        else:
            # The supplied form contained errors - just print them to the terminal.
            print (form.errors)
    else:
        # If the request was not a POST, display the form to enter details.
        form = CategoryForm()

    # Bad form (or form details), no form supplied...
    # Render the form with error messages (if any).
    return render(request, 'dravate/add_category.html', {'form': form})
```

References

- Python, <http://www.python.org>
- Pip, <http://www.pip-installer.org>
- Django, <https://www.djangoproject.com>
- Git, <http://git-scm.com>
- GitHub, <https://github.com>
- HTML, <http://www.w3.org/html/>
- CSS, <http://www.w3.org/Style/CSS/>
- JQuery, <http://jquery.com>
- Twitter Bootstrap, <http://getbootstrap.com/>
- Bing Search API via Azure Datamarket, <http://datamarket.azure.com>
- PythonAnywhere, <https://www.pythonanywhere.com>