

DJANGO FRAMEWORK



Guizani Ahmed

Django project

1- create virtual environment with python3 **venv**:

```
PS C:\Users\Ahmed\django> python -m venv django-venv
```

2- activate the venv:

```
PS C:\Users\Ahmed\django> django-venv\scripts\activate
```

Rq (pour sortir de venv)

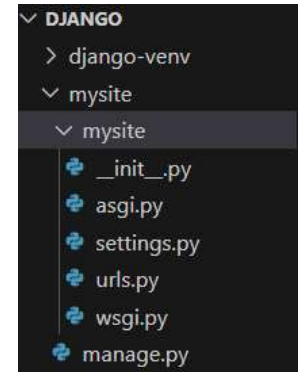
```
(django-venv) PS C:\Users\Ahmed\django> deactivate
```

3- install DJANGO (installation dans le venv seulement):

```
(django-venv) PS C:\Users\Ahmed\django> pip install django
```

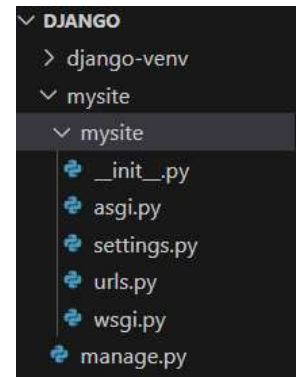
4- Creating a project:

```
(django-venv) PS C:\Users\Ahmed\django> django-admin startproject mysite
```



Django project:

- **mysite/** root directory is a container for your project.
- **manage.py**: A command-line utility that lets you interact with this Django project in various ways.
- **mysite/___init___.py**: An empty file that tells Python that this directory should be considered a Python package.
- **manage.py** is automatically created in each Django project. It does the same thing as **django-admin** but also sets the **DJANGO_SETTINGS_MODULE** environment variable so that it points to your project's **settings.py** file.



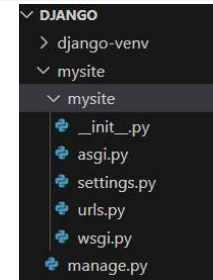
```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

Django project: settings.py

- **mysite/settings.py**: Settings/configuration for this Django project.



```
"""
Django settings for mysite project.

Generated by 'django-admin startproject' using Django 5.0.1.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.0/ref/settings/
"""

from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-64n=5_&qf&wdw2xn52j0lavoibf)$c%$!&c66785maw0#~v$51'

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

ALLOWED_HOSTS = []
```

```
# Application definition

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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'mysite.urls'
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]

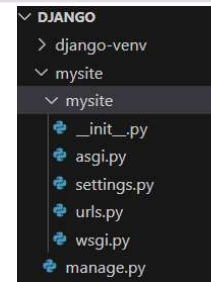
WSGI_APPLICATION = 'mysite.wsgi.application'

# Database
# https://docs.djangoproject.com/en/5.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Django project: settings.py

- `mysite/settings.py`: Settings/configuration for this Django project.



```
# Password validation
# https://docs.djangoproject.com/en/5.0/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/5.0/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.0/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

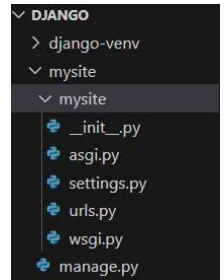
Django project: `urls.py`

- **`mysite/urls.py`**: The URL declarations for this Django project; a “table of contents” of your Django-powered site.

```
"""
URL configuration for mysite project.

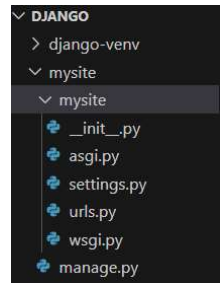
The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/5.0/topics/http/urls/
Examples:
Function views
    1. Add an import: from my_app import views
    2. Add a URL to urlpatterns: path('', views.home, name='home')
Class-based views
    1. Add an import: from other_app.views import Home
    2. Add a URL to urlpatterns: path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path

urlpatterns = [
    path('admin/', admin.site.urls),
]
```



Django project: `asgi.py` / `wsgi.py`

- **`mysite/asgi.py`**: An entry-point for ASGI-compatible web servers to serve your project.
- **`mysite/wsgi.py`**: An entry-point for WSGI-compatible web servers to serve your project.



```
"""
WSGI config for mysite project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')

application = get_wsgi_application()
```

```
"""
ASGI config for mysite project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/5.0/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')

application = get_asgi_application()
```


The development server

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

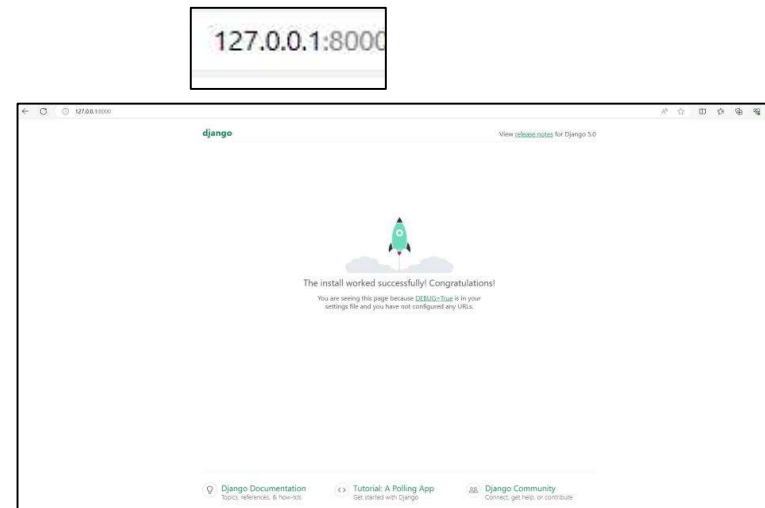
You have 18 unapplied migration(s). Your project may not work properly until
Run 'python manage.py migrate' to apply them.
January 19, 2024 - 10:02:42
Django version 5.0.1, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[19/Jan/2024 10:02:54] "GET / HTTP/1.1" 200 10629
Not Found: /favicon.ico
[19/Jan/2024 10:02:54] "GET /favicon.ico HTTP/1.1" 404 2110
```

Rq (pour sortir de run server): ctrl + c

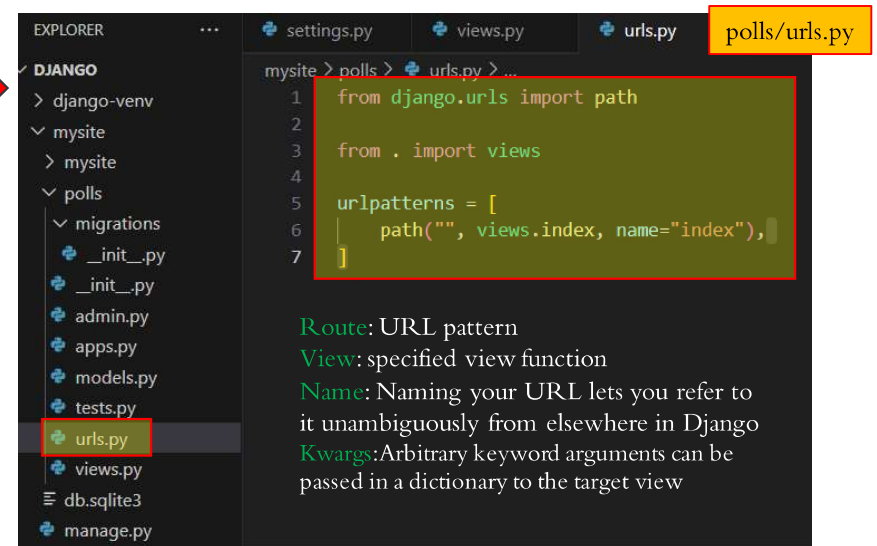
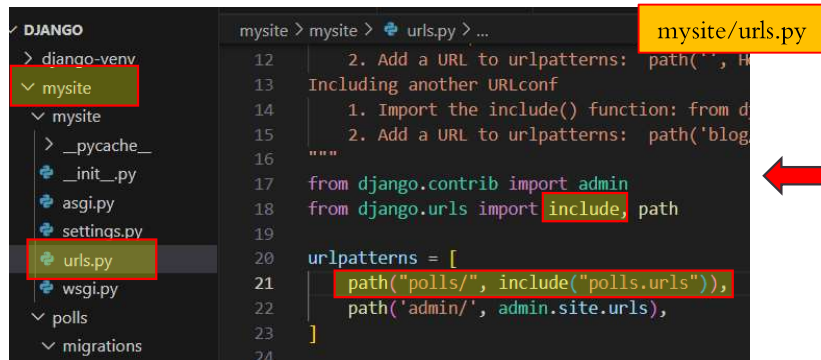
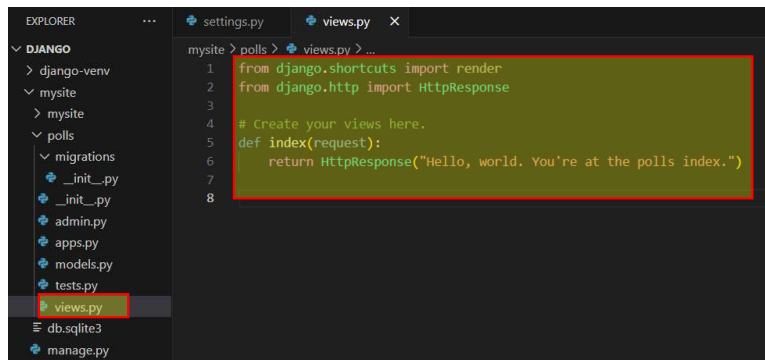
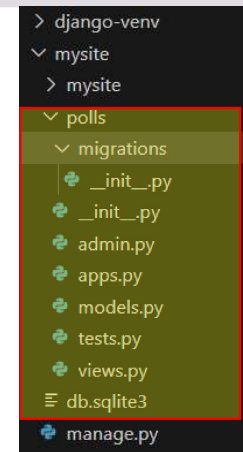
Autre paramètre pour activer le serveur:

`python manage.py runserver 0.0.0.0:8000`



Creating the Polls app

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py startapp polls
```



```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py runserver
```



Database setup

- Mysite/settings.py.
- By default, the configuration uses SQLite.
- you may want to use a more scalable database like PostgreSQL, to avoid database-switching headaches down the road
- If you wish to use another database, install the appropriate [database bindings](#) and change the following keys in the **DATABASES** 'default' item to match your database connection settings:
 - **ENGINE** Either `'django.db.backends.sqlite3'`, `'django.db.backends.postgresql'`, `'django.db.backends.mysql'`, or `'django.db.backends.oracle'`.
 - **NAME** – The name of your database. If you're using SQLite, the database will be a file on your computer; in that case, **NAME** should be the full absolute path, including filename, of that file. The default value, **BASE_DIR / 'db.sqlite3'**, will store the file in your project directory.
- If you are not using SQLite as your database, additional settings such as **USER**, **PASSWORD**, and **HOST** must be added. For more details, see the reference documentation for **DATABASES**.

Database setup

- Mysite/settings.py.

```
TIME_ZONE = 'UTC+1'
```

```
# Application definition

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

By default, `INSTALLED_APPS` contains the following apps, all of which come with Django:

- [`django.contrib.admin`](#) – The admin site. You'll use it shortly.
- [`django.contrib.auth`](#) – An authentication system.
- [`django.contrib.contenttypes`](#) – A framework for content types.
- [`django.contrib.sessions`](#) – A session framework.
- [`django.contrib.messages`](#) – A messaging framework.
- [`django.contrib.staticfiles`](#) – A framework for managing static files.

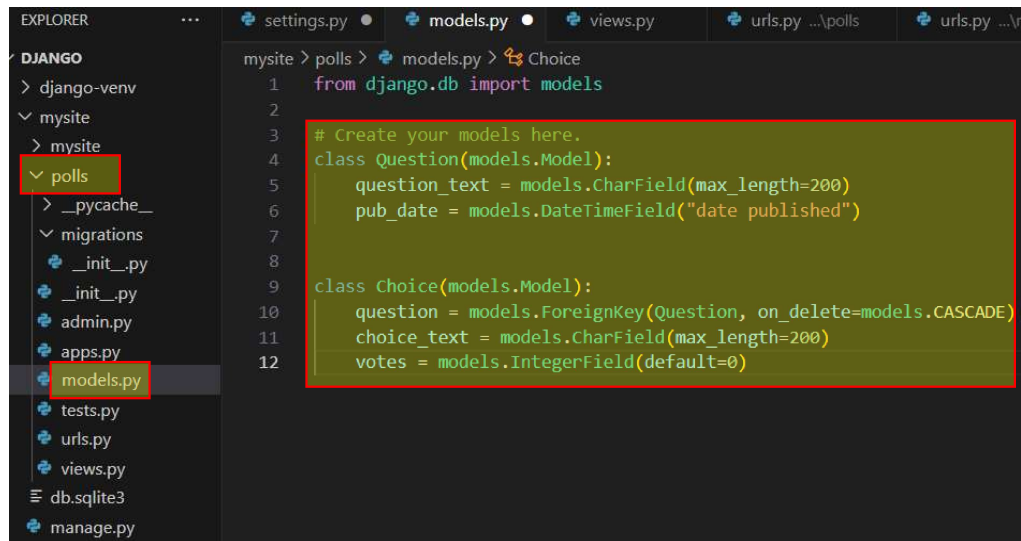
we need to create the tables in the database before we can use them. To do that, run the following command:

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py migrate
```

The `migrate` command looks at the `INSTALLED_APPS` setting and creates any necessary database tables according to the database settings in your `mysite/settings.py` file and the database migrations shipped with the app

Creating models

- `polls/models.py`.
- In our poll app, we'll create two models: **Question** and **Choice**. A **Question** has a question and a publication date. A **Choice** has two fields: the text of the choice and a vote tally. Each **Choice** is associated with a **Question**.



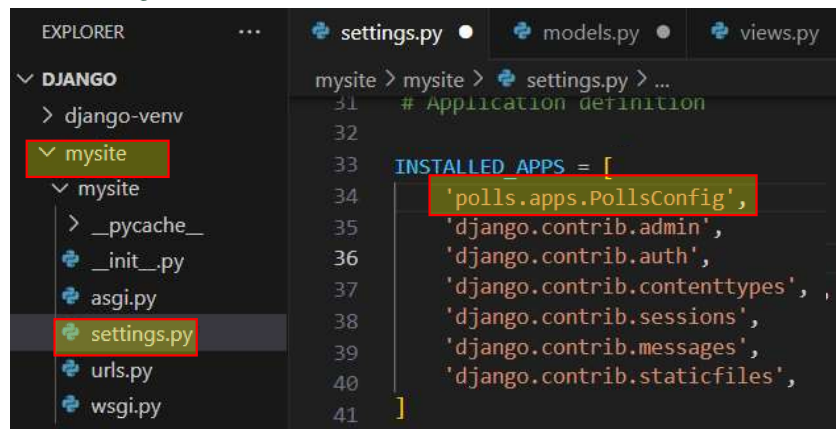
```
EXPLORER
DJANGO
> django-venv
  > mysite
    > mysite
      > polls
        > __pycache__
        migrations
        __init__.py
        __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        urls.py
        views.py
        db.sqlite3
        manage.py

mysite > polls > models.py > Choice
1  from django.db import models
2
3  # Create your models here.
4  class Question(models.Model):
5      question_text = models.CharField(max_length=200)
6      pub_date = models.DateTimeField("date published")
7
8
9  class Choice(models.Model):
10     question = models.ForeignKey(Question, on_delete=models.CASCADE)
11     choice_text = models.CharField(max_length=200)
12     votes = models.IntegerField(default=0)
```

- `CharField` for character fields and `DateTimeField` for datetimes. This tells Django what type of data each field holds.

Activating models

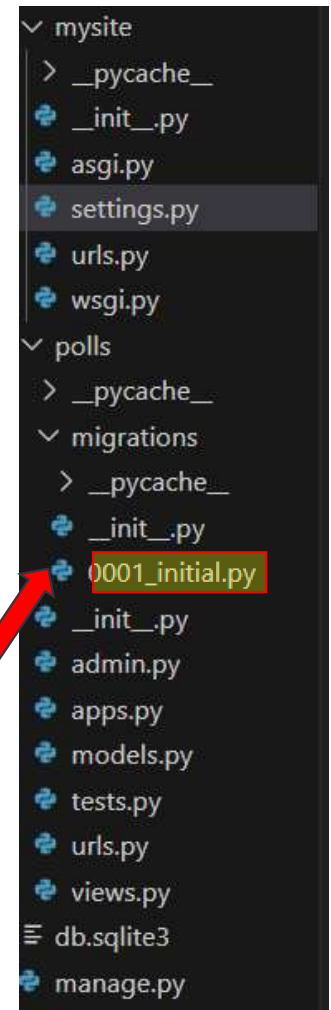
- `mysite/settings.py`.
- That small bit of model code gives Django a lot of information. With it, Django is able to:
 - Create a database schema (CREATE TABLE statements) for this app.
 - Create a Python database-access API for accessing Question and Choice objects.
- But first we need to tell our project that the polls app is installed, To include the app in our project, we need to add a reference to its configuration class in the `INSTALLED_APPS` setting. The `PollsConfig` class is in the `polls/apps.py` file, so its dotted path is `'polls.apps.PollsConfig'`.



```
EXPLORER
mysite
  > __pycache__
  > __init__.py
  > asgi.py
  > settings.py
  > urls.py
  > wsgi.py

mysite > mysite > settings.py > ...
31 # Application definition
32
33 INSTALLED_APPS = [
34     'polls.apps.PollsConfig',
35     'django.contrib.admin',
36     'django.contrib.auth',
37     'django.contrib.contenttypes',
38     'django.contrib.sessions',
39     'django.contrib.messages',
40     'django.contrib.staticfiles',
41 ]
```

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py makemigrations polls
Migrations for 'polls':
  polls\migrations\0001_initial.py
    - Create model Question
    - Create model Choice
```



```
mysite
  > __pycache__
  > __init__.py
  > asgi.py
  > settings.py
  > urls.py
  > wsgi.py
polls
  > __pycache__
  > __init__.py
  > 0001_initial.py
  > __init__.py
  > admin.py
  > apps.py
  > models.py
  > tests.py
  > urls.py
  > views.py
db.sqlite3
manage.py
```

To read migration for the
model

running `makemigrations`,
you're telling Django that
you've made some
changes to your models

Activating models

- There's a command that will run the migrations for you and manage your database schema automatically - that's called migrate, and we'll come to it in a moment - but first, let's see what SQL that migration would run. The sqlmigrate command takes migration names and returns their SQL:

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py sqlmigrate polls 0001
BEGIN;
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "question_text" varchar(200) NOT NULL, "pub_date" datetime NOT NULL);
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT NULL, "question_id" bigint NOT NULL REFERENCES "polls_question" ("id") DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice" ("question_id");
COMMIT;
```

- Now, run migrate again to create those model tables in your database

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK
```

- python manage.py makemigrations to create migrations for those changes
- python manage.py migrate to apply those changes to the database.

Playing with the API

- let's hop into the interactive Python shell and play around with the free API Django gives you. To invoke the Python shell, use this command:

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py shell
```

```
>>> from polls.models import Choice, Question # Import the model classes we just wrote.

# No questions are in the system yet.
>>> Question.objects.all()
<QuerySet []>

# Create a new Question.
# Support for time zones is enabled in the default settings file, so
# Django expects a datetime with tzinfo for pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.
>>> q.save()

# Now it has an ID.
>>> q.id
1

# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217, tzinfo=datetime.timezone.utc)

# Change values by changing the attributes, then calling save().
>>> q.question_text = "What's up?"
>>> q.save()

# objects.all() displays all the questions in the database.
>>> Question.objects.all()
<QuerySet [<Question: Question object (1)>]>
```


Playing with the API

Wait a minute. `<Question: Question object (1)>` isn't a helpful representation of this object. Let's fix that by editing the **Question** model (in the `polls/models.py` file) and adding a `__str__()` method to both **Question** and **Choice**:

```
polls/models.py 1

from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

It's important to add `__str__()` methods to your models, not only for your own convenience when dealing with the interactive prompt, but also because objects' representations are used throughout Django's automatically-generated admin.

Let's also add a custom method to this model:

```
polls/models.py 1

import datetime

from django.db import models
from django.utils import timezone

class Question(models.Model):
    # ...
    def was_published_recently(self):
        return self.pub_date >= timezone.now() - datetime.timedelta(days=1)
```

Creating an admin user

```
python manage.py createsuperuser
```

```
Username (leave blank to use 'ahmed'): admin
Email address: admin@example.com
Password:
Password (again):
```

Start the development server

```
(django-venv) PS C:\Users\Ahmed\django\mysite> python manage.py runserver
```

127.0.0.1:8000/admin/login/?next=/admin/

Django administration

Username:

Password:

Log in

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

+ Add

Change

Users

+ Add

Change

Recent actions

My actions

None available

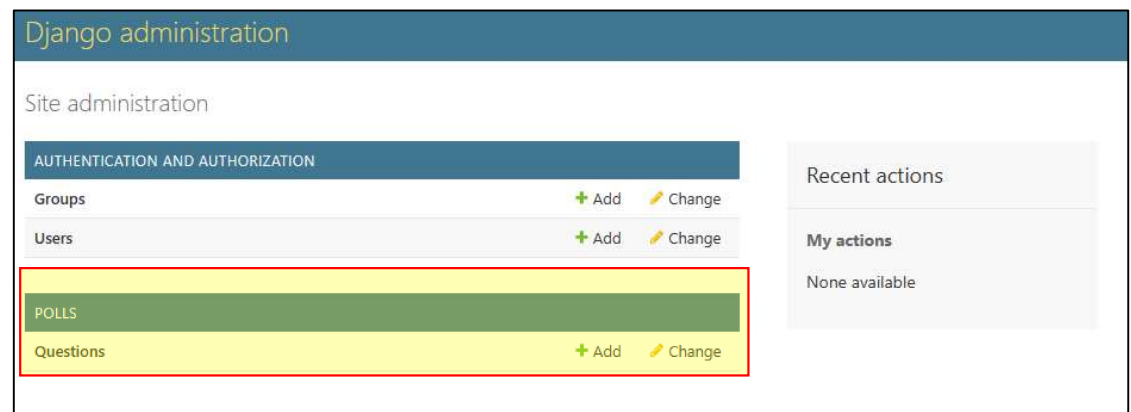
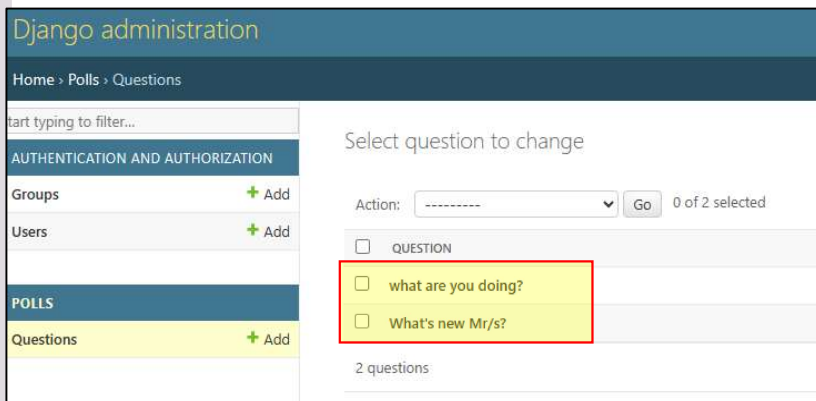
Creating an admin user

Make the poll app modifiable in the admin

Only one more thing to do: we need to tell the admin that Question objects have an admin interface. To do this, open the `polls/admin.py` file, and edit it to look like this:

```
DJANGO
> django-venv
v mysite
> mysite
v polls
  > __pycache__
  v migrations
    > __pycache__
    _init_.py
    0001_initial.py
  _init_.py
  admin.py
  apps.py

mysite > polls > admin.py
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Question
5
6  admin.site.register(Question)
```



Design admin framework

polls/admin.py file, and edit it to look like this:

```
settings.py  index.html  urls.py  views.py  admin.py  re

mysite > polls > admin.py > ...
1  from django.contrib import admin
2
3  # Register your models here.
4  from .models import Question
5  class QuestionAdmin(admin.ModelAdmin):
6      fieldsets = [
7          (None, {"fields": ["question_text"]}),
8          ("Date information", {"fields": ["pub_date"]}),
9      ]
10
11
12  admin.site.register(Question, QuestionAdmin)
13
```

Home > Polls > Questions > what are you doing?

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups [+ Add](#)

Users [+ Add](#)

POLLS

Questions [+ Add](#)

Change question

what are you doing?

Question text:

Date information

Date published: Date: Today |

Time: Now |

Note: You are 1 hour ahead of server time.

[SAVE](#) [Save and add another](#) [Save and continue editing](#)

Creating more views

Now let's add a few more views to `polls/views.py`. These views are slightly different, because they take an argument:

```
polls/views.py

def detail(request, question_id):
    return HttpResponse("You're looking at question %s." % question_id)

def results(request, question_id):
    response = "You're looking at the results of question %s."
    return HttpResponse(response % question_id)

def vote(request, question_id):
    return HttpResponse("You're voting on question %s." % question_id)
```

Wire these new views into the `polls.urls` module by adding the following `path()` calls:

```
polls/urls.py

from django.urls import path

from . import views

urlpatterns = [
    # ex: /polls/
    path("", views.index, name="index"),
    # ex: /polls/5/
    path("<int:question_id>/", views.detail, name="detail"),
    # ex: /polls/5/results/
    path("<int:question_id>/results/", views.results, name="results"),
    # ex: /polls/5/vote/
    path("<int:question_id>/vote/", views.vote, name="vote"),
]
```

```
polls/views.py

from django.http import HttpResponse

from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by("-pub_date")[:5]
    output = ", ".join([q.question_text for q in latest_question_list])
    return HttpResponse(output)

# Leave the rest of the views (detail, results, vote) unchanged
```

There's a problem here, though: the page's design is hard-coded in the view. If you want to change the way the page looks, you'll have to edit this Python code. So let's use Django's template system to separate the design from Python by creating a template that the view can use.

First, create a directory called `templates` in your `polls` directory. Django will look for templates in there.

Your project's `TEMPLATES` setting describes how Django will load and render templates. The default settings file configures a `DjangoTemplates` backend whose `APP_DIRS` option is set to `True`. By convention `DjangoTemplates` looks for a "templates" subdirectory in each of the `INSTALLED_APPS`.

Within the `templates` directory you have just created, create another directory called `polls`, and within that create a file called `index.html`. In other words, your template should be at `polls/templates/polls/index.html`. Because of how the `app_directories` template loader works as described above, you can refer to this template within Django as `polls/index.html`.

```
polls/templates/polls/index.html

{% if latest_question_list %}
    <ul>
    {% for question in latest_question_list %}
        <li><a href="/polls/{{ question.id }}">{{ question.question_text }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>No polls are available.</p>
{% endif %}
```

Creating more views

A shortcut: `render()`

It's a very common idiom to load a template, fill a context and return an `HttpResponse` object with the result of the rendered template. Django provides a shortcut. Here's the full `index()` view, rewritten:

```
polls/views.py 1

from django.shortcuts import render

from .models import Question

def index(request):
    latest_question_list = Question.objects.order_by("-pub_date")[:5]
    context = {"latest_question_list": latest_question_list}
    return render(request, "polls/index.html", context)
```

Note that once we've done this in all these views, we no longer need to import `loader` and `HttpResponse` (you'll want to keep `HttpResponse` if you still have the stub methods for `detail`, `results`, and `vote`).

The `render()` function takes the request object as its first argument, a template name as its second argument and a dictionary as its optional third argument. It returns an `HttpResponse` object of the given template rendered with the given context.

Creating more views

A shortcut: `get_object_or_404()` ¶

It's a very common idiom to use `get()` and raise `Http404` if the object doesn't exist. Django provides a shortcut. Here's the `detail()` view, rewritten:

```
polls/views.py ¶

from django.shortcuts import get_object_or_404, render

from .models import Question

# ...
def detail(request, question_id):
    question = get_object_or_404(Question, pk=question_id)
    return render(request, "polls/detail.html", {"question": question})
```

The `get_object_or_404()` function takes a Django model as its first argument and an arbitrary number of keyword arguments, which it passes to the `get()` function of the model's manager. It raises `Http404` if the object doesn't exist.



Philosophy

Why do we use a helper function `get_object_or_404()` instead of automatically catching the `ObjectDoesNotExist` exceptions at a higher level, or having the model API raise `Http404` instead of `ObjectDoesNotExist`?

Because that would couple the model layer to the view layer. One of the foremost design goals of Django is to maintain loose coupling. Some controlled coupling is introduced in the `django.shortcuts` module.

There's also a `get_list_or_404()` function, which works just as `get_object_or_404()` – except using `filter()` instead of `get()`. It raises `Http404` if the list is empty.

Create a test to expose to bug

What we've just done in the `shell` to test for the problem is exactly what we can do in an automated test, so let's turn that into an automated test.

A conventional place for an application's tests is in the application's `tests.py` file; the testing system will automatically find tests in any file whose name begins with `test`.

Put the following in the `tests.py` file in the `polls` application:

```
polls/tests.py

import datetime

from django.test import TestCase
from django.utils import timezone

from .models import Question

class QuestionModelTests(TestCase):
    def test_was_published_recently_with_future_question(self):
        """
        was_published_recently() returns False for questions whose pub_date
        is in the future.
        """
        time = timezone.now() + datetime.timedelta(days=30)
        future_question = Question(pub_date=time)
        self.assertIs(future_question.was_published_recently(), False)
```

Here we have created a `django.test.TestCase` subclass with a method that creates a `Question` instance with a `pub_date` in the future. We then check the output of `was_published_recently()` - which *ought* to be False.

Create a test to expose to bug

Running tests

In the terminal, we can run our test:

```
$ python manage.py test polls
```

and you'll see something like:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
-----
FAIL: test_was_published_recently_with_future_question (polls.tests.QuestionModelTests)
-----
Traceback (most recent call last):
  File "/path/to/mysite/polls/tests.py", line 16, in
test_was_published_recently_with_future_question
    self.assertIs(future_question.was_published_recently(), False)
AssertionError: True is not False
-----

Ran 1 test in 0.001s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

[LINK1: WWW.UDEMY.COM/COURSE/DJANGO-OFFICIAL-TUTORIAL-2-2/LEARN/LECTURE/14475880/#LEARNING-TOOLS](https://www.udemy.com/course/django-official-tutorial-2-2/learn/lecture/14475880/#learning-tools)

[LINK 2 : DOCS.DJANGOPROJECT.COM/EN/5.0/INTRO/TUTORIAL01/](https://docs.djangoproject.com/en/5.0/intro/tutorial01/)

Fixing the bug

Fixing the bug

We already know what the problem is: `Question.was_published_recently()` should return **False** if its `pub_date` is in the future. Amend the method in `models.py`, so that it will only return **True** if the date is also in the past:

polls/models.py 11

```
def was_published_recently(self):
    now = timezone.now()
    return now - datetime.timedelta(days=1) <= self.pub_date <= now
```

and run the test again:

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
```

.

```
-----
Ran 1 test in 0.001s
```

OK

```
Destroying test database for alias 'default'...
```