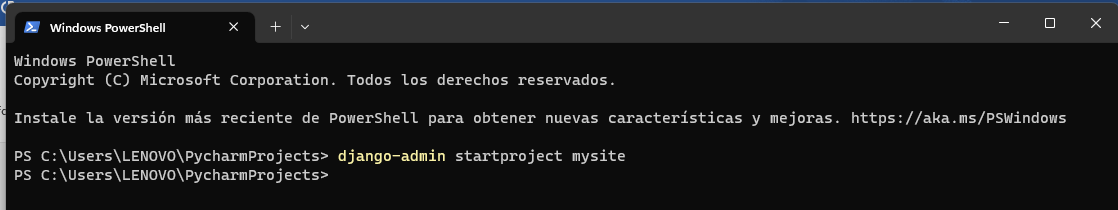
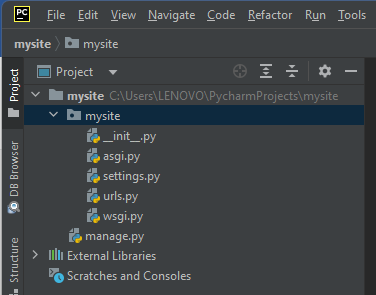
Crear un proyecto en Django



En CMD de Windows



Abro el proyecto en PyCharm



These files are:

The outer **mysite/** root directory is a container for your project. Its name doesn’t matter to Django; you can rename it to anything you like.

**manage.py:** A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin and manage.py.

The inner **mysite/** directory is the actual Python package for your project. Its name is the Python package name you’ll need to use to import anything inside it (e.g. mysite.urls).

**mysite/\_\_init\_\_.py:** An empty file that tells Python that this directory should be considered a Python package. If you’re a Python beginner, read more about packages in the official Python docs.

**mysite/settings.py:** Settings/configuration for this Django project. Django settings will tell you all about how settings work.

**mysite/urls.py:** The URL declarations for this Django project; a “table of contents” of your Django-powered site. You can read more about URLs in URL dispatcher.

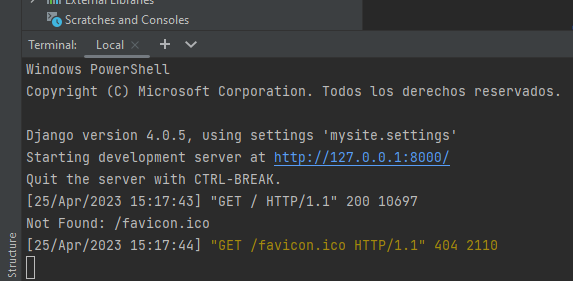
**mysite/asgi.py:** An entry-point for ASGI-compatible web servers to serve your project. See How to deploy with ASGI for more details.

**mysite/wsgi.py:** An entry-point for WSGI-compatible web servers to serve your project. See How to deploy with WSGI for more details.

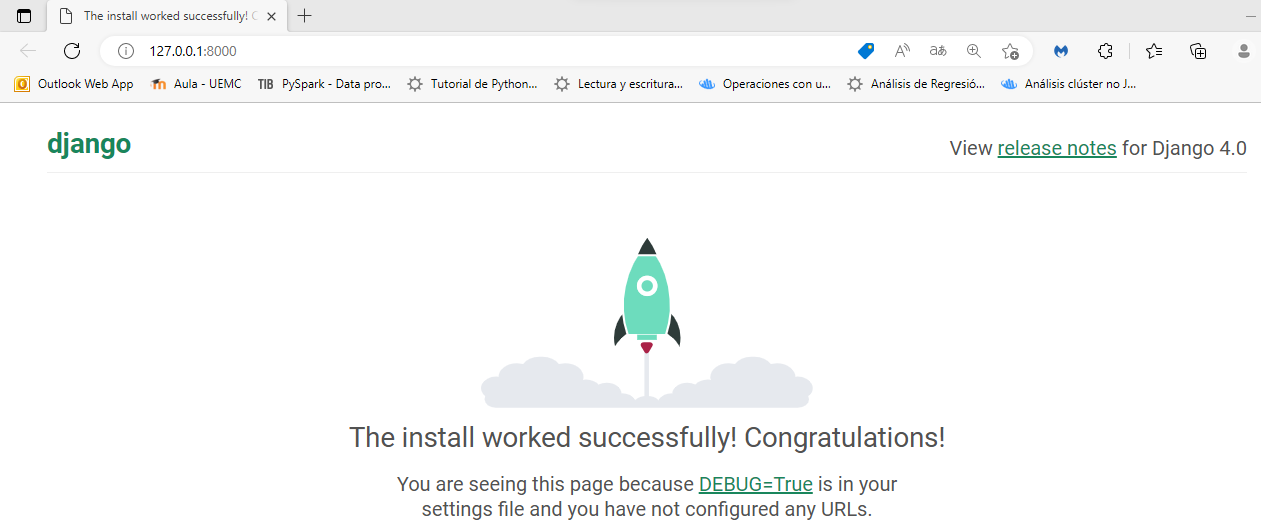
Corremos y arrancamos el server



En la terminal de la aplicación en PyCharm

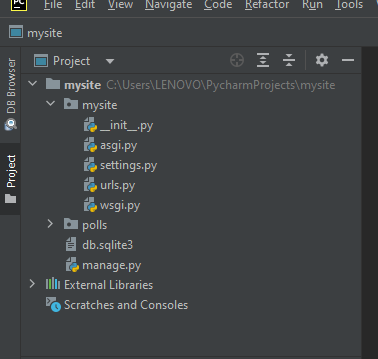


Y en el navegador…

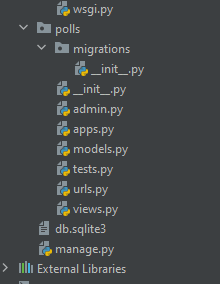


Para crear una app dentro de la carpeta del proyecto tipeamos el siguiente comando en el CMD





Para generar una view abrimos un file views.py en la app



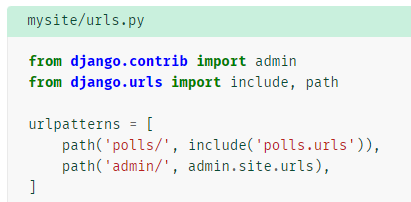
Y Codeamos



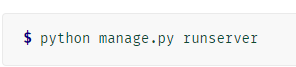
Para llamar la vista necesitamos mapearla en una URL. Creamos el file en la app urls.py y codeamos



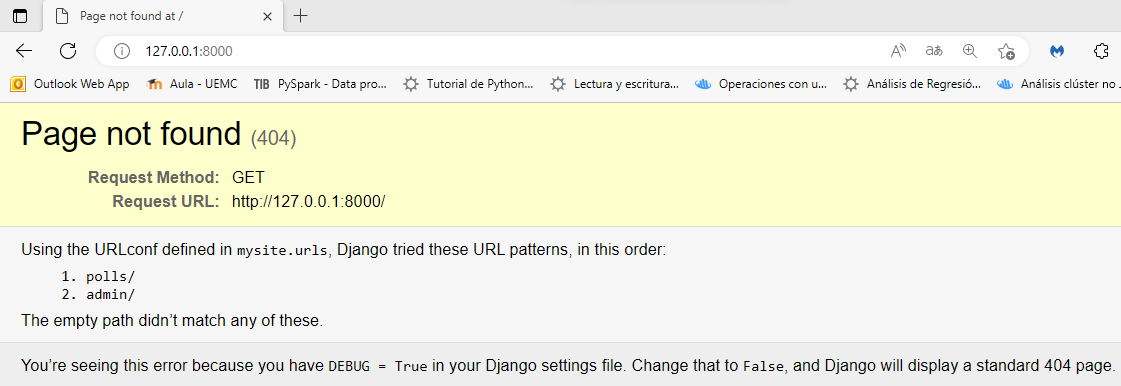
El siguiente paso es apuntar la raíz URLconf al módulo polls.urls. En mysite/urls.py, agregue una importación para django.urls.include e inserte un include() en la lista de patrones de URL, de modo que tenga:



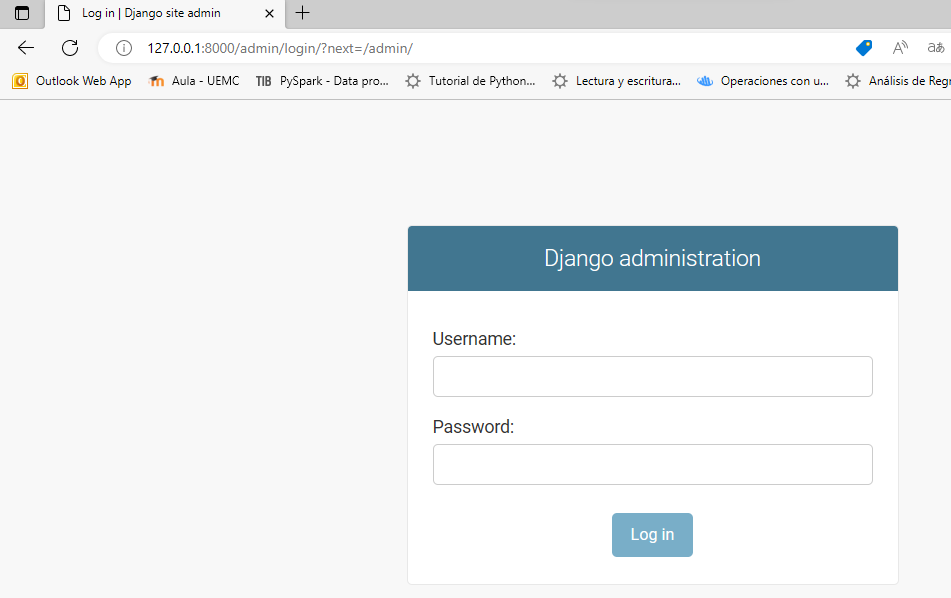
Arrancamos el servidor

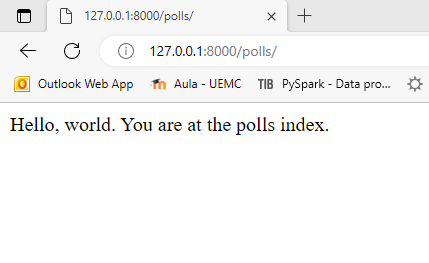


Y en el navegador…

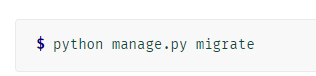


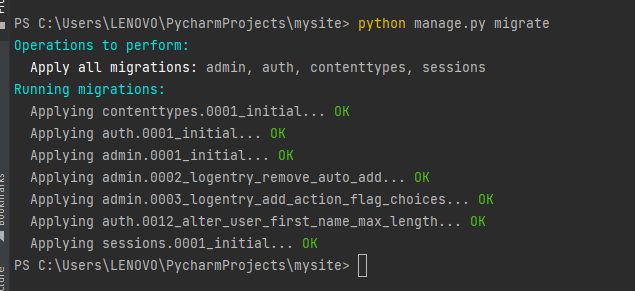
Y si completamos la ruta con admin o polls(s/ definición de la url)



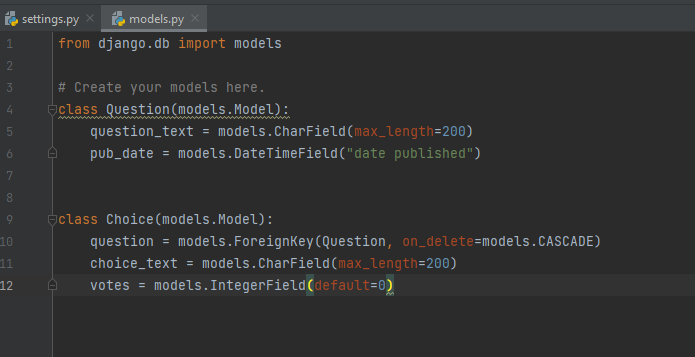


Migramos las aplicaciones del proyecto (ver en setting.py installed app) y se generan las tablas necesarias

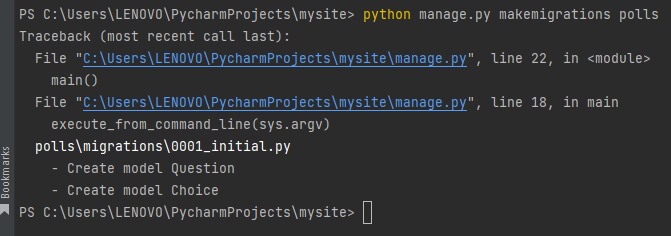




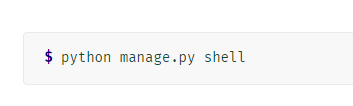
Creamos el modelo de datos en la app en polls/models.py



Activamos el modelo



Para explorar



**>>> 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)>]>

Save these changes and start a new Python interactive shell by running **python manage.py shell** again:

**>>> from** **polls.models** **import** Choice, Question

# Make sure our \_\_str\_\_() addition worked.

**>>>** Question.objects.all()

<QuerySet [<Question: What's up?>]>

# Django provides a rich database lookup API that's entirely driven by

# keyword arguments.

**>>>** Question.objects.filter(id=1)

<QuerySet [<Question: What's up?>]>

**>>>** Question.objects.filter(question\_text\_\_startswith="What")

<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.

**>>> from** **django.utils** **import** timezone

**>>>** current\_year = timezone.now().year

**>>>** Question.objects.get(pub\_date\_\_year=current\_year)

<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.

**>>>** Question.objects.get(id=2)

Traceback (most recent call last):

...

DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a

# shortcut for primary-key exact lookups.

# The following is identical to Question.objects.get(id=1).

**>>>** Question.objects.get(pk=1)

<Question: What's up?>

# Make sure our custom method worked.

**>>>** q = Question.objects.get(pk=1)

**>>>** q.was\_published\_recently()

True

# Give the Question a couple of Choices. The create call constructs a new

# Choice object, does the INSERT statement, adds the choice to the set

# of available choices and returns the new Choice object. Django creates

# a set to hold the "other side" of a ForeignKey relation

# (e.g. a question's choice) which can be accessed via the API.

**>>>** q = Question.objects.get(pk=1)

# Display any choices from the related object set -- none so far.

**>>>** q.choice\_set.all()

<QuerySet []>

# Create three choices.

**>>>** q.choice\_set.create(choice\_text="Not much", votes=0)

<Choice: Not much>

**>>>** q.choice\_set.create(choice\_text="The sky", votes=0)

<Choice: The sky>

**>>>** c = q.choice\_set.create(choice\_text="Just hacking again", votes=0)

# Choice objects have API access to their related Question objects.

**>>>** c.question

<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.

**>>>** q.choice\_set.all()

<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

**>>>** q.choice\_set.count()

3

# The API automatically follows relationships as far as you need.

# Use double underscores to separate relationships.

# This works as many levels deep as you want; there's no limit.

# Find all Choices for any question whose pub\_date is in this year

# (reusing the 'current\_year' variable we created above).

**>>>** Choice.objects.filter(question\_\_pub\_date\_\_year=current\_year)

<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking again>]>

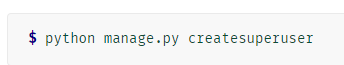
# Let's delete one of the choices. Use delete() for that.

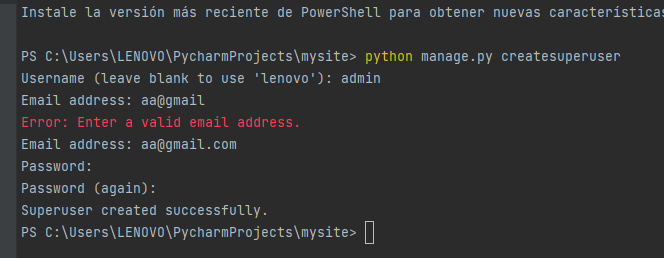
**>>>** c = q.choice\_set.filter(choice\_text\_\_startswith="Just hacking")

**>>>** c.delete()

Introduccion a Django Admin

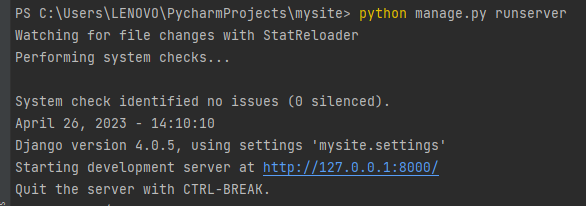
Crear super usuario para acceder al Admin

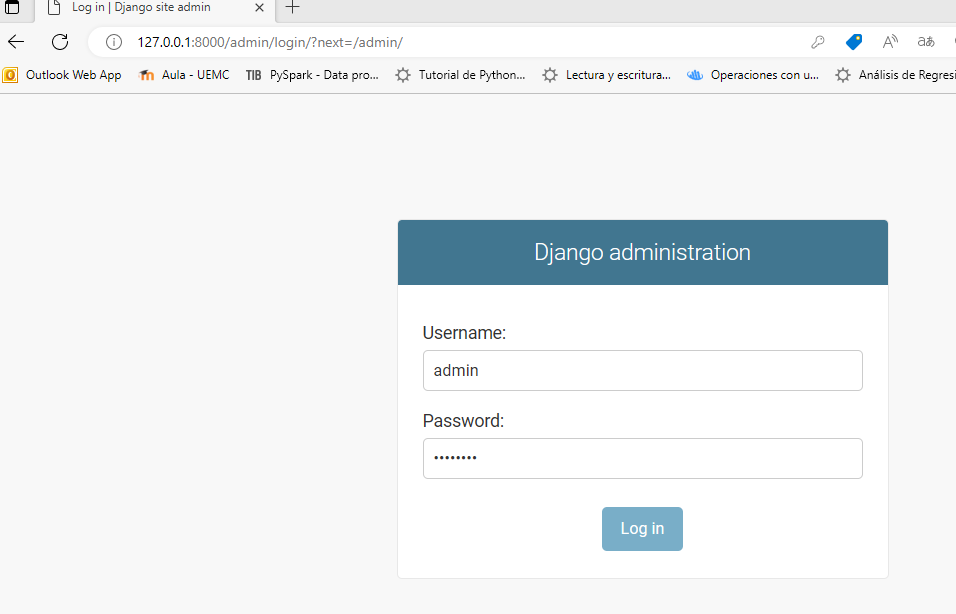


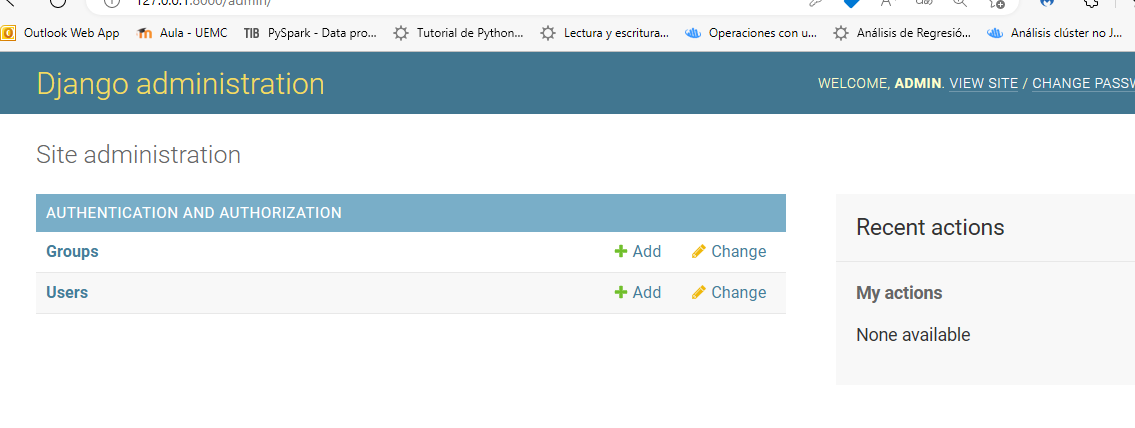


Activamos el server

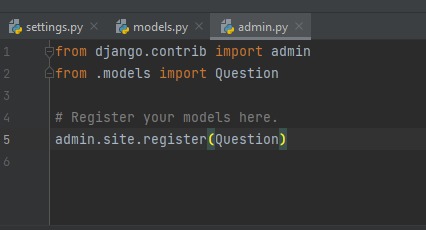


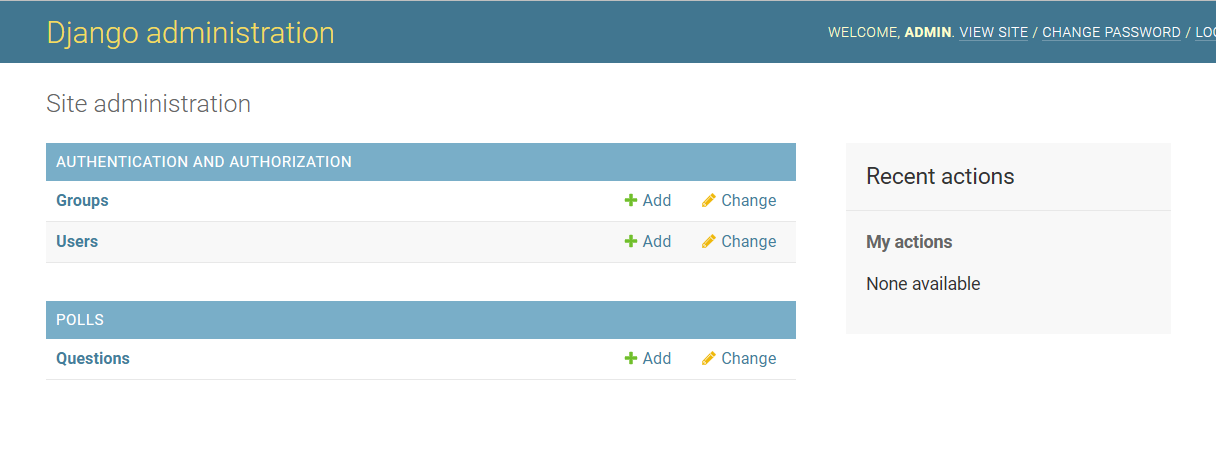




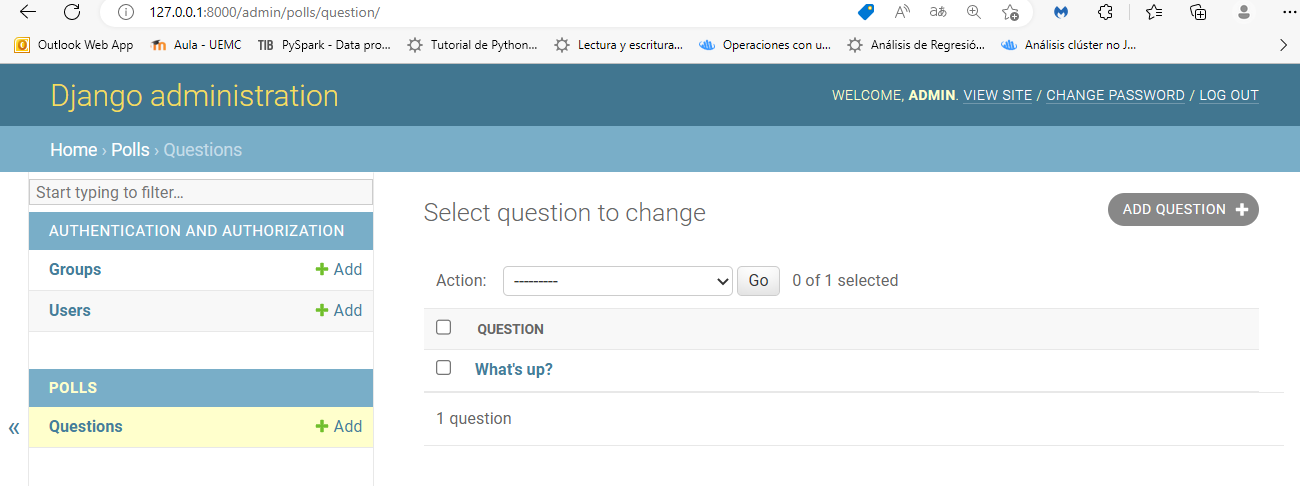


Para ver y administrar el modelo tenemos que registrarlo en **polls/admin.py**

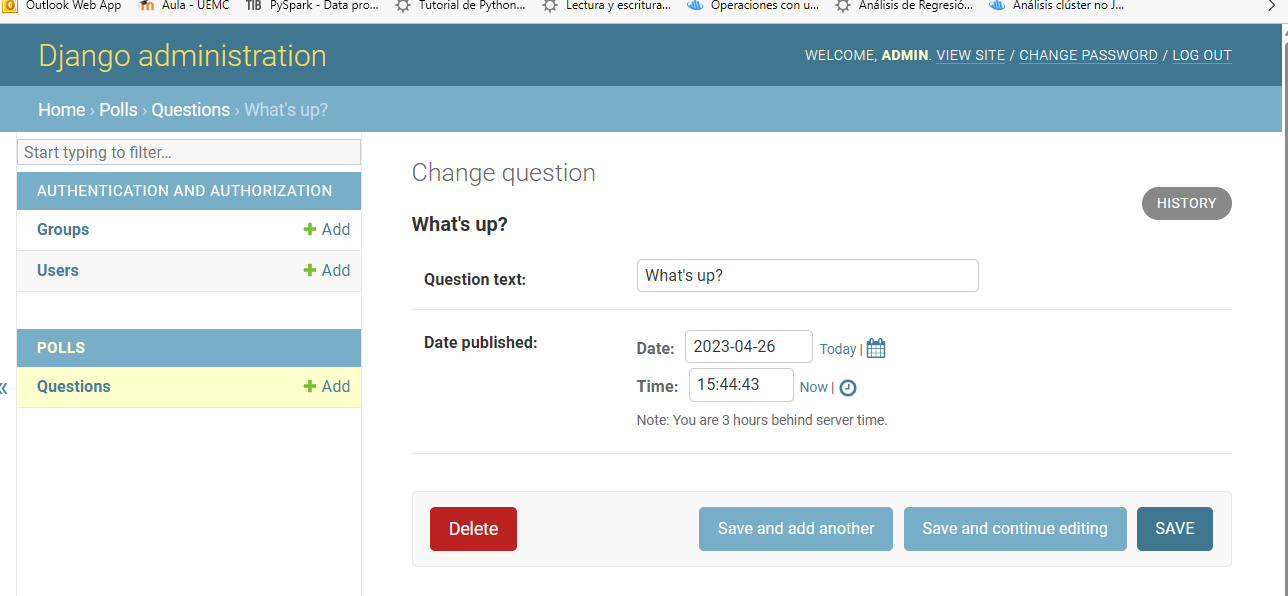




Click en el modelo de la app y … podemos administrar

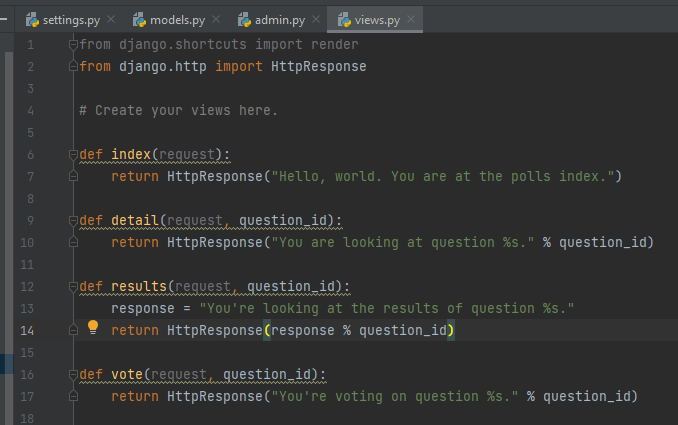


Y si vamos al dato



Agregar vistas (views)

En polls/views.py codificamos



En polls.urls module agregamos los path() para los calls

