SASA NI PYTHON

Explanation of the basic files

https://docs.djangoproject.com/en/5.2/intro/tutorial01/

Steps to Apply the Configuration

Save settings.py: Update the DATABASES setting with your MySQL details.

Test the Connection: Run the following command to verify Django can connect to the MySQL database:

bash

Copy

python manage.py check

If there are no errors, the connection is successful.

GREATING A NEW APP INSIDE MY PROJECT:

Use:

python manage.py startapp mydreams

Eg.Run python manage.py startapp mydrams in your project directory (e.g., mysite/) creates the following structure:

SETTING UP DATABASE AND ADDING APPS TO SETTING FILE

https://docs.djangoproject.com/en/5.2/intro/tutorial02/

THIS FILE IS IMPORTANT:

$ python manage.py makemigrations polls

You should see something similar to the following:

Migrations for 'polls':

polls/migrations/0001\_initial.py

+ Create model Question

+ Create model Choice

By running makemigrations, you’re telling Django that you’ve made some changes to your models (in this case, you’ve made new ones)

and that you’d like the changes to be stored as a migration.

PROCESS OF MAKING MIGRATIONS

Run python manage.py makemigrations to generate migration files for the polls app. This should create a migration file (e.g., polls/migrations/0001\_initial.py) that defines the Question and Choice tables.

Run python manage.py migrate to apply the migrations to the MySQL database.

Expected Output: After running makemigrations, you should see a new migration file in polls/migrations/.

After migrate, tables like polls\_question and polls\_choice should appear in MySQL.

Common Issue: If you previously used a different database (e.g., SQLite)

and switched to MySQL, old migration files may cause conflicts.

Delete the polls/migrations/ directory (except \_\_init\_\_.py) and

the db.sqlite3 file (if present), then re-run makemigrations and migrate.

THE ORDER OF MAKING MIGRATIONS IS AS FOLLOWS:

python manage.py makemigrations

FOLLOWED BY: python manage.py migrate

POINT TO NOTE:

Migrations are how Django stores changes to your models (and thus your database schema) - they’re files on disk. You can read the migration for your new model if you like; it’s the file **polls/migrations/0001\_initial.py**.

POINT TO NOTE:

if you’re interested, you can also run [**python manage.py check**](https://docs.djangoproject.com/en/5.2/ref/django-admin/#django-admin-check); this checks for any problems in your project without making migrations or touching the database.

GREATING AN ADMIN:

**Creating an admin user**[**¶**](https://docs.djangoproject.com/en/5.2/intro/tutorial02/#creating-an-admin-user)

First we’ll need to create a user who can login to the admin site. Run the following command:

/ 

**$** python manage.py createsuperuser

Enter your desired username and press enter.

Username: admin

You will then be prompted for your desired email address:

Email address: [admin@example.com](mailto:admin@example.com)

PASSWORD: admin

**Make the poll app modifiable in the admin**[**¶**](https://docs.djangoproject.com/en/5.2/intro/tutorial02/#make-the-poll-app-modifiable-in-the-admin)

But where’s our poll app? It’s not displayed on the admin index page.

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:

polls/admin.py[**¶**](https://docs.djangoproject.com/en/5.2/intro/tutorial02/#id6)

**from** **django.contrib** **import** admin

**from** **.models** **import** Question

admin.site.register(Question)

WHEN DISPLAYING CONTENT IN HTML PAGE, USE TEMPLATE

**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()**](https://docs.djangoproject.com/en/5.2/topics/http/shortcuts/#django.shortcuts.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()**](https://docs.djangoproject.com/en/5.2/ref/models/querysets/#django.db.models.query.QuerySet.get) function of the model’s manager. It raises [**Http404**](https://docs.djangoproject.com/en/5.2/topics/http/views/#django.http.Http404) if the object doesn’t exist.

## **Namespacing URL names**[¶](https://docs.djangoproject.com/en/5.2/intro/tutorial03/#namespacing-url-names)

The tutorial project has just one app, **polls**. In real Django projects, there might be five, ten, twenty apps or more. How does Django differentiate the URL names between them? For example, the **polls** app has a **detail** view, and so might an app on the same project that is for a blog. How does one make it so that Django knows which app view to create for a url when using the **{% url %}** template tag?

The answer is to add namespaces to your URLconf. In the **polls/urls.py** file, go ahead and add an **app\_name** to set the application namespace:

polls/urls.py[**¶**](https://docs.djangoproject.com/en/5.2/intro/tutorial03/#id12)

**from** **django.urls** **import** path

**from** **.** **import** views

app\_name = "polls"

urlpatterns = [

path("", views.index, name="index"),

path("<int:question\_id>/", views.detail, name="detail"),

path("<int:question\_id>/results/", views.results, name="results"),

path("<int:question\_id>/vote/", views.vote, name="vote"),

]

EXAMPLE:

<**li**><**a** href="{% **url** 'polls:detail' question.id %}">{{ question.question\_text }}</**a**></**li**>

POINT TO NOTE,FOR YOUR APP TO WORK AND APPEAR ON ADMIN DASHBOARD,IT MUST BE ADDED TO THE REGISTERED TO ADMIN.PY AS FOLLOWS

from django.contrib import admin

# Register your models here.

from .models import MyDream

admin.site.register(MyDream)