

Preparing environment

<code>mkdir project_name && cd \$_</code>	Create project folder and navigate to it
<code>python -m venv env_name</code>	Create venv for the project
<code>source env_name\bin\activate</code>	Activate environment (Replace "bin" by "Scripts" in Windows)
<code>pip install django</code>	Install Django (and others dependencies if needed)
<code>pip freeze > requirements.txt</code>	Create requirements file
<code>pip install -r requirements.txt</code>	Install all required files based on your pip freeze command
<code>git init</code>	Version control initialisation, be sure to create appropriate gitignore

Create project

<code>django-admin startproject mysite (or I like to call it config)</code>	This will create a mysite directory in your current directory the manage.py file
<code>python manage.py runserver</code>	You can check that everything went fine

Database Setup

<code>Open up mysite/settings.py</code>	It's a normal Python module with module-level variables representing Django settings.
<code>ENGINE – 'django.db.backends.sqlite3', 'django.db.backends.postgresql', 'django.db.backends.mysql', or 'django.db.backends.oracle'</code>	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
<code>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.</code>	The default value, <code>BASE_DIR / 'db.sqlite3'</code> , will store the file in your project directory.
<code>If you are not using SQLite as your database, additional settings such as USER, PASSWORD, and HOST must be added.</code>	For more details, see the reference documentation for DATABASES.

Creating an app

<code>python manage.py startapp app_name</code>	Create an app_name directory and all default file/folder inside
<code>INSTALLED_APPS = ['app_name', ...</code>	Apps are "pluggable", that will "plug in" the app into the project
<code>urlpatterns = [path('app_name/', include('app_name.urls')), path('admin/', admin.site.urls),]</code>	Into urls.py from project folder, include app urls to project



By **Olivier R. (OGR)**
cheatography.com/ogr/

Published 6th February, 2022.
 Last updated 12th February, 2022.
 Page 1 of 8.

Sponsored by **ApolloPad.com**
 Everyone has a novel in them. Finish Yours!
<https://apollopad.com>

Creating models

<code>class ModelName(models.Model):</code>	Create your class in the <code>app_name/models.py</code> file
<code>title = models.CharField(max_length=100)</code>	Create your fields
<code>def __str__(self): return self.title</code>	It's important to add <code>__str__()</code> methods to your models, because objects' representations are used throughout Django's automatically-generated admin.

Database editing

<code>python manage.py makemigrations (app_name)</code>	By running <code>makemigrations</code> , you're telling Django that you've made some changes to your models
<code>python manage.py sqlmigrate #identifier</code>	See what SQL that migration would run.
<code>python manage.py check</code>	This checks for any problems in your project without making migrations
<code>python manage.py migrate</code>	Create those model tables in your database
<code>python manage.py shell</code>	Hop into the interactive Python shell and play around with the free API Django gives you

Administration

<code>python manage.py createsuperuser</code>	Create a user who can login to the admin site
<code>admin.site.register(ModelName)</code>	Into <code>app_name/admin.py</code> , add the model to administration site
<code>http://127.0.0.1:8000/admin/</code>	Open a web browser and go to <code>"/admin/"</code> on your local domain

Management

<code>mkdir app_name/management app_name/management/commands && cd \$_</code>	Create required folders
<code>touch your_command_name.py</code>	Create a python file with your command name
<code>from django.core.management.base import BaseCommand #import anything else you need to work with (models?)</code>	Edit your new python file, start with <code>import</code>
<code>class Command(BaseCommand): help = "This message will be shown with the --help option after your command" def handle(self, args, *kwargs): # Work the command is supposed to do</code>	Create the <code>Command</code> class that will handle your command
<code>python manage.py my_custom_command</code>	And this is how you execute your custom command

Django lets you create your custom CLI commands



By **Olivier R. (OGR)**
cheatography.com/ogr/

Published 6th February, 2022.
Last updated 12th February, 2022.
Page 2 of 8.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
<https://apollopod.com>

Write your first view

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("Hello, world. You're at the
index.")
```

Open the file `app_name/views.py` and put the following Python code in it.

This is the simplest view possible.

```
from django.urls import path
from . import views
```

In the `app_name/urls.py` file include the following code.

```
app_name = "app_name"
urlpatterns = [
    path('', views.index, name='index'),
]
```

View with argument

```
def detail(request, question_id):
    return HttpResponse(f"You're looking at question {question_id}")
```

Exemple of view with an arugment

```
urlpatterns = [
    path('<int:question_id>/', views.detail, name='detail'),
    ...
```

See how we pass argument in path

```
{% url 'app_name:view_name' question_id %}
```

We can pass attribute from template this way

View with Template

```
app_name/templates/app_name/index.html
```

This is the folder path to follow for template

```
context = {'key': value}
```

Pass values from view to template

```
return render(request, 'app_name/index.html', context)
```

Exemple of use of render shortcut

```
{% Code %}
```

Edit template with those. Full list here

```
{{ Variavle from view's context dict }}
```

```
<a href="{% url 'detail' question.id %}"></a>
```

```
<title>Page Title</title>
```

you can put this on top of your html template to define page title

Add some static files

```
'django.contrib.staticfiles'
```

Be sure to have this in your INSTALLED-
_APPS

```
STATIC_URL = 'static/'
```

The given exemples are for this config

```
mkdir app_name/static app_name/static/app_name
```

Create static folder associated with your
app

```
{% load static %}
```

Put this on top of your template

```
<link rel="stylesheet" type="text/css" href="{% static 'app_name/st-
yle.css' %}">
```

Exemple of use of static.



Raising 404

```
raise Http404("Question does not exist")
```

in a try / except statement

```
question = get_object_or_404(Question, pk=question_id)
```

A shortcut

Forms

```
app_name/forms.py
```

Create your form classes here

```
from django import forms
```

Import django's forms module

```
from .models import YourModel
```

import models you need to work with

```
class ExempleForm(forms.Form):
```

For very simple forms, we can use simple Form class

```
    exemple_field = forms.CharField(label='E-
    xemple label', max_length=100)
```

```
class ExempleForm(forms.ModelForm):
```

A ModelForm maps a model class's fields to HTML form <input> elements via a Form. Widget is optional. Use it to override default widget

```
    class meta:
        model = model_name
        fields = ["fields"]
        labels = {"text": "label_text"}
        widget = {"text": forms.widget_name}
```

```
TextInput, EmailInput, PasswordInput,
DateInput, Textarea
```

Most common widget list

```
if request.method != "POST":
    form = ExempleForm()
```

Create a blank form if no data submitted

```
form = ExempleForm(data=request.POST)
```

The form object contain's the informations submitted by the user

```
is form.isvalid()
```

Form validation. Always use redirect function

```
    form.save()
    return redirect("app_name:view_name",
argument=ardument)
```

```
{% csrf_token %}
```

Template tag to prevent "cross-site request forgery" attack

Render Form In Template

```
{{ form.as_p }}
```

The most simple way to render the form, but usually it's ugly

```
{{ field|placeholder:field.label }}
```

The | is a filter, and here for placeholder, it's a custom one. See next section to see how to create it

```
{{ form.username|placeholder:"Your name here"}}
```

```
{% for field in form %}
{{form.username}}
```

You can extract each fields with a for loop.
Or by explicitly specifying the field



Custom template tags and filters

<code>app_name\templatetags__init__.py</code>	Create this folder and this file. Leave it blank
<code>app_name\templatetags\filter_name.py</code>	Create a python file with the name of the filter
<code>{% load filter_name %}</code>	Add this on top of your template
<code>from django import template</code>	To be a valid tag library, the module must contain a module-level variable named <code>register</code>
<code>register = template.Library()</code>	that is a <code>template.Library</code> instance
<code>@register.filter(name='cut')</code> <code>def cut(value, arg):</code> <code>"""Removes all values of arg from the given string"""</code> <code>return value.replace(arg, '')</code>	Here is an exemple of filter definition. See the decorator? It registers your filter with your <code>Library</code> instance. You need to restart server for this to take effects
https://tech.serhatteker.com/post/2021-06/placeholder-template-tags/	Here is a link of how to make a placeholder custom template tag

Setting Up User Accounts

Create a "users" app	Don't forget to add app to settings.py and include the URLs from users.
<code>app_name = "users"</code> <code>urlpatterns[</code> <code># include default auth urls.</code> <code>path("", include("django.contrib.auth.urls"))</code> <code>]</code>	Inside <code>app_name/urls.py</code> (create it if inexistent), this code includes some default authentication URLs that Django has defined.
<code>{% if form.error %}</code> <code><p>Your username and password didn't match</p></code> <code>{% endif %}</code> <code><form method="post" action="{% url 'users:login' %}"></code> <code>{% csrf_token %}</code> <code>{{ form.as_p }}</code> <code><button name="submit">Log in</button></code> <code><input type="hidden" name="next" value="{% url 'app_name:index' %}" /></code> <code></form></code>	Basic login.html template Save it at save template as <code>users/templates/registration/login.html</code> We can access to it by using <code>Log in</code>
<code>{% if user.is_authenticated %}</code>	Check if user is logged in
<code>{% url "users:logout" %}</code>	Link to logout page, and log out the user save template as <code>users/templates/registration/logged_out.html</code>
<code>path("register/", views.register, name="register"),</code>	Inside <code>app_name/urls.py</code> , add path to register



By **Olivier R. (OGR)**
cheatography.com/ogr/

Published 6th February, 2022.
Last updated 12th February, 2022.
Page 5 of 8.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish Yours!
<https://apollopad.com>

Setting Up User Accounts (cont)

```

from django.shortcuts import render, redirect
from django.contrib.auth import login
from django.contrib.forms import UserCreationForm

def register(request):
    if request.method != "POST":
        form = UserCreationForm()
    else:
        form = UserCreationForm(data=request.POST)

    if form.is_valid():
        new_user = form.save()
        login(request, new_user)
        return redirect("app_name:index")

    context = {"form": form}
    return render(request, "registration/registration.html", context)

```

We write our own register() view inside users/views.py
 For that we use UserCreationForm, a django building model.
 If method is not post, we render a blank form
 Else, is the form pass the validity check, an user is created
 We just have to create a registration.html template in same folder as the login and logged_out

Allow Users to Own Their Data

```

...
from django.contrib.auth.decorators import
login_required
...

@login_required
def my_view(request)
    ...

...
from django.contrib.auth.models import User
...
owner = models.ForeignKey(User, on_delete=models.CASCADE)

user_data = ExempleModel.objects.filter(owner=request.user)

...
from django.http import Http404
...

...
if exemple_data.owner != request.user:
    raise Http404

```

Restrict access with @login_required decorator

If user is not logged in, they will be redirect to the login page
 To make this work, you need to modify settings.py so Django knows where to find the login page

Add the following at the very end

```

# My settings
LOGIN_URL = "users:login"

```

Add this field to your models to connect data to certain users

When migrating, you will be prompt to select a default value

Use this kind of code in your views to filter data of a specific user
 request.user only exist when user is logged in

Make sure the data belongs to the current user

If not the case, we raise a 404



Allow Users to Own Their Data (cont)

<code>new_data = form.save(commit=False)</code>	Don't forget to associate user to your data in corresponding views
<code>new_data.owner = request.user</code>	
<code>new_data.save()</code>	The "commit=False" attribute let us do that

Paginator

<code>from django.core.paginator import Paginator</code>	In <code>app_name/views.py</code> , import Paginator
<code>exemple_list = Exemple.objects.all()</code>	In your class view, Get a list of data
<code>paginator = Paginator(exemple_list, 5) # Show 5 items per page.</code>	Set appropriate pagination
<code>page_number = request.GET.get('page')</code>	Get actual page number
<code>page_obj = paginator.get_page(page_number)</code>	Create your Page Object, and put it in the context
<code>{% for item in page_obj %}</code>	The Page Object acts now like your list of data
<pre><div class="pagination"> {% if page_obj.has_previous %} &laquo; first previous {% endif %} Page {{ page_obj.number }} of {{ page_obj.paginator.num_pages }}. {% if page_obj.has_next %} next last &raquo; {% endif %} </div></pre>	An exemple of what to put on the bottom of your page to navigate through Page Objects

Deploy to Heroku

<code>https://heroku.com</code>	Make a Heroku account
<code>https://devcenter.heroku.com/articles/heroku-cli/</code>	Install Heroku CLI
<pre>pip install psychog2 pip install django-heroku pip install gunicorn</pre>	install these packages
<code>pip freeze > requirements.txt</code>	update requirements.txt



Deploy to Heroku (cont)

```
# Heroku settings.  
import django_heroku  
django_heroku.settings(locals(), staticfiles=False)  
  
if os.environ.get('DEBUG') == "TRUE":  
    DEBUG = True  
  
elif os.environ.get('DEBUG') == "FALSE":  
    DEBUG = False
```

At the very end of settings.py, make an Heroku settings section
import django_heroku and tell django to apply django heroku settings
The staticfiles to false is not a viable option in production, check whitenoise for that IMO

C

By **Olivier R. (OGR)**
cheatography.com/ogr/

Published 6th February, 2022.
Last updated 12th February, 2022.
Page 8 of 8.

Sponsored by **ApolloPad.com**
Everyone has a novel in them. Finish
Yours!
<https://apollopad.com>