Cheatography

Django Cheat Sheet

by Olivier R. (OGR) via cheatography.com/143343/cs/30794/

Preparing environnement

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

Create project

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

Database Setup

Open up `mysite/settings.py`	It's a normal Python module with module-level variables representing Django settings.
`ENGINE` – `'django.db.backends.sqlite3'`, `'django.db.backends.postgresql'`, `'django.db.backends.mysql'`, or `'django.db.backends.oracle'`	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
`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.

Creating an app

`python manage.py startapp app_name`	Create an app_name directory and all default file/folder inside

`INSTALLED_APPS = [`
`'app_name',`
`...`
	Apps are "plugable", that will "plug in" the app into the project

`urlpatterns = [`
`path('app_name/', include('app_name.urls')),`
`path('admin/', admin.site.urls),`
`]`
	Into urls.py from project folder, inculde app urls to project

Cheatography

Django Cheat Sheet

by Olivier R. (OGR) via cheatography.com/143343/cs/30794/

Creating models

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

Database editing

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

Administration

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

Management

`mkdir app_name/management app_name/management/commands && cd $_`	Create required folders
`touch your_command_name.py`	Create a python file with your command name
`from django.core.management.base import BaseCommand` `#import anything else you need to work with (models?)`	Edit your new python file, start with import

`class Command(BaseCommand):`
`  help = "This message will be shon with the --help option after your command"`
 	
`  def handle(self, *args*, *kwargs):`
`  # Work the command is supposed to do`
	Create the Command class that will handle your command

`python manage.py my_custom_command`	And this is how you execute your custom command

Django lets you create your customs CLI commands

Write your first view

Open the file app_name/views.py and put the following Python code in it.
This is the simplest view possible.

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

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

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

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

View with argument

Exemple of view with an arugment

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

See how we pass argument in path

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

We can pass attribute from template this way

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

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 %}`
`{{ Variavle from view's context dict }}`
`<a href="{% url 'detail' question.id %}"></a>`
	Edit template with those. Full list here

`<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/style.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):` `  exemple_field = forms.CharField(label='E-xemple label', max_length=100)`	For very simple forms, we can use simple Form class

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

`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.save()`
`  return redirect("app_name:view_name", argument=ardument)`
	Form validation. Always use redirect function

`{% 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 usualy it's ugly
`{{ field|placeholder:field.label }}` `{{ form.username|placeholder:"Your name here"}}`	The | is a filter, and here for placeholder, it's a custom one. See next section to see how to create it
`{% 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

`app_name\templatetags\__init__.py`	Create this folder and this file. Leave it blank
`app_name\templatetags\filter_name.py`	Create a python file with the name of the filter
`{% load filter_name %}`	Add this on top of your template

`from django import template`
 	
`register = template.Library()`
	To be a valid tag library, the module must contain a module-level variable named register
	that is a template.Library instance

`@register.filter(name='cut')`
`def cut(value, arg):`
`  """Removes all values of arg from the given string"""`
`  return value.replace(arg, '')`
	Here is an exemple of filter definition.
	See the decorator? It registers your filter with your Library instance.
	You need to restart server for this to take effects

https://tech.serhatteker.com/post/2021-06/placeholder-templa-tetags/	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.

`app_name = "users"`
`urlpatterns[`
`  # include default auth urls.`
`  path("", include("django.contribe.auth.urls"))`
`]`
	Inside app_name/urls.py (create it if inexistent),
	this code includes some default authentification URLs that Django has defined.

`{% if form.error %}`
`  <p>Your username and password didn't match</p>`
`{% endif %}`
`<form method="post" action="{% url 'users:login' %}">`
`  {% csrf_token %}`
`  {{ form.as_p }}`
 	
`  <button name="submit">Log in</button>`
`  <input type="hidden" name="next" value=" {% url 'app_n-ame:index' %}" />`
`</form>`
	Basic login.html template
	Save it at save template as
	users/templates/registration/login.html
	We can access to it by using
	`<a href="{% url 'users:login' %}">Log in</a>`

`{% if user.is_authenticated %}`	Check if user is logged in
`{% url "users:logout" %}`	Link to logout page, and log out the user save template as users/templates/registration/logged_out.html
`path("register/", views.register, name="register"),`	Inside app_name/urls.py, add path to register

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/regis-
ter.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)
  ...
```

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"
```

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

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

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

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

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

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

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

Make sure the data belongs to the current user

If not the case, we raise a 404

Allow Users to Own Their Data (cont)

```
new_data = form.save(commit=false)
new_data.owner = request.user
new_data.save()
```

Don't forget to associate user to your data in corresponding views

The "commit=false" attribute let us do that

Paginator

`from django.core.paginator import Paginator`	In app_name/views.py, import Paginator
`exemple_list = Exemple.objects.all()`	In your class view, Get a list of data
`paginator = Paginator(exemple_list, 5) # Show 5 items per page.`	Set appropriate pagination
`page_number = request.GET.get('page')`	Get actual page number
`page_obj = paginator.get_page(page_number)`	Create your Page Object, and put it in the context
`{% for item in page_obj %}`	The Page Object acts now like your list of data

An exemple of what to put on the bottom of your page
to navigate through Page Objects

```
<div class="pagination">
  <span class="step-links">
  {% if page_obj.has_previous %}
    <a href="?page=1">&laquo; first</a>
    <a href="?page={{ page_obj.previous_page_number }}">previous</a>
  {% endif %}
    <span class="current"> Page {{ page_obj.number }} of {{ page_obj.pag-
inator.num_pages }}. </span>
  {% if page_obj.has_next %}
   <a href="?page={{ page_obj.next_page_number }}">next</a>
   <a href="?page={{ page_obj.paginator.num_pages }}">last &raquo;</a>
   {% endif %}
   </span>
</div>
```

Deploy to Heroku

https://heroku.com	Make a Heroku account
https://devcenter.heroku.com:articles/heroku-cli/	Install Heroku CLI

`pip install psycog2`
`pip install django-heroku`
`pip install gunicorn`
	install these packages

`pip freeze > requirements.txt`	updtate requirements.txt

Deploy to Heroku (cont)

```
# Heroku settings.
import django_heroku
django_heroku.settings(locals(), staticfil-
es=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 ettings 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

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

Published 6th February, 2022.
Last updated 12th February, 2022.