**Django model**

A model in Django is a special kind of object – it is saved in the database. A database is a collection of data. This is a place in which you will store information about users, your blog posts, etc. We will be using a SQLite database to store our data. This is the default Django database adapter – it'll be enough for us right now.

You can think of a model in the database as a spreadsheet with columns (fields) and rows (data).

### Creating a blog post model

In the blog/models.py file we define all objects called Models – this is a place in which we will define our blog post.

Let's open blog/models.py in the code editor, remove everything from it, and write code like this:

blog/models.py

from django.conf import settings

from django.db import models

from django.utils import timezone

class Post(models.Model):

author = models.ForeignKey(settings.AUTH\_USER\_MODEL, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

text = models.TextField()

created\_date = models.DateTimeField(default=timezone.now)

published\_date = models.DateTimeField(blank=True, null=True)

def publish(self):

self.published\_date = timezone.now()

self.save()

def \_\_str\_\_(self):

return self.title

Double-check that you use two underscore characters (\_) on each side of str. This convention is used frequently in Python and sometimes we also call them "dunder" (short for "double-underscore").

It looks scary, right? But don't worry – we will explain what these lines mean!

All lines starting with from or import are lines that add some bits from other files. So instead of copying and pasting the same things in every file, we can include some parts with from ... import ....

class Post(models.Model): – this line defines our model (it is an object).

* class is a special keyword that indicates that we are defining an object.
* Post is the name of our model. We can give it a different name (but we must avoid special characters and whitespace). Always start a class name with an uppercase letter.
* models.Model means that the Post is a Django Model, so Django knows that it should be saved in the database.

Now we define the properties we were talking about: title, text, created\_date, published\_dateand author. To do that we need to define the type of each field (Is it text? A number? A date? A relation to another object, like a User?)

* models.CharField – this is how you define text with a limited number of characters.
* models.TextField – this is for long text without a limit. Sounds ideal for blog post content, right?
* models.DateTimeField – this is a date and time.
* models.ForeignKey – this is a link to another model.

### Create tables for models in your database

The last step here is to add our new model to our database. First we have to make Django know that we have some changes in our model. (We have just created it!) Go to your console window and type python manage.py makemigrations blog. It will look like this:

command-line

(myvenv) ~/djangogirls$ python manage.py makemigrations blog

Migrations for 'blog':

blog/migrations/0001\_initial.py:

- Create model Post

**Note:** Remember to save the files you edit. Otherwise, your computer will execute the previous version which might give you unexpected error messages.

Django prepared a migration file for us that we now have to apply to our database. Type python manage.py migrate blog and the output should be as follows:

command-line

(myvenv) ~/djangogirls$ python manage.py migrate blog

Operations to perform:

Apply all migrations: blog

Running migrations:

Applying blog.0001\_initial... OK

# Django admin

To add, edit and delete the posts we've just modeled, we will use Django admin.

Let's open the blog/admin.py file in the code editor and replace its contents with this:

blog/admin.py

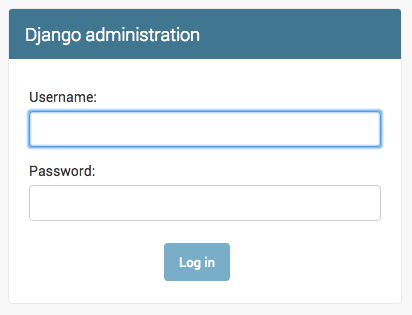
from django.contrib import admin

from .models import Post

admin.site.register(Post)

As you can see, we import (include) the Post model defined in the previous chapter. To make our model visible on the admin page, we need to register the model with admin.site.register(Post).

OK, time to look at our Post model. Remember to run python manage.py runserver in the console to run the web server. Go to your browser and type the address http://127.0.0.1:8000/admin/. You will see a login page like this:



To log in, you need to create a superuser - a user account that has control over everything on the site. Go back to the command line, type python manage.py createsuperuser, and press enter.

Remember, to write new commands while the web server is running, open a new terminal window and activate your virtualenv. We reviewed how to write new commands in the **Your first Django project!** chapter, in the **Starting the web server** section.

Mac OS X or Linux:

(myvenv) ~/djangogirls$ python manage.py createsuperuser

Windows:

(myvenv) C:\Users\Name\djangogirls> python manage.py createsuperuser

When prompted, type your username (lowercase, no spaces), email address, and password. **Don't worry that you can't see the password you're typing in – that's how it's supposed to be.** Type it in and press enter to continue. The output should look like this (where the username and email should be your own ones):

Username: ola

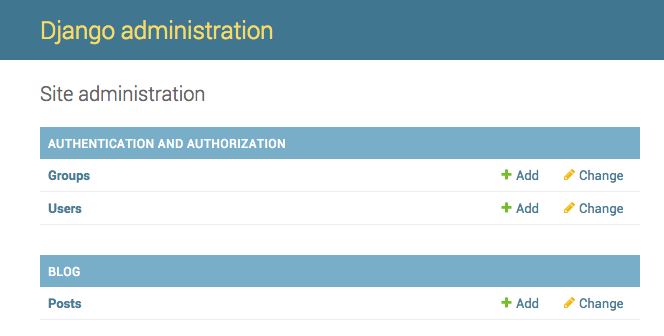
Email address: ola@example.com

Password:

Password (again):

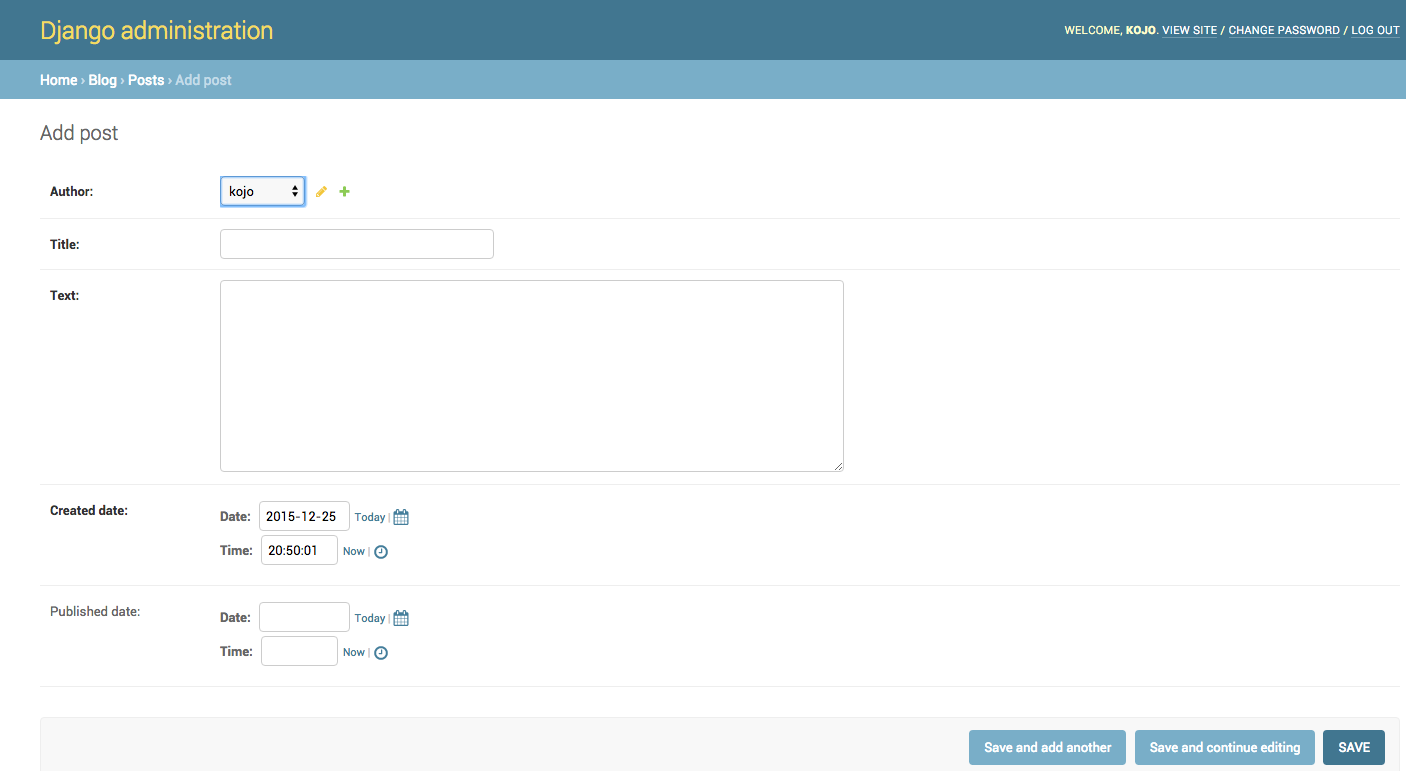
Superuser created successfully.

Return to your browser. Log in with the superuser's credentials you chose; you should see the Django admin dashboard.



Go to Posts and experiment a little bit with it. Add five or six blog posts. Don't worry about the content –- it's only visible to you on your local computer -- you can copy-paste some text from this tutorial to save time. :)

Make sure that at least two or three posts (but not all) have the publish date set. It will be helpful later.



If you want to know more about Django admin, you should check Django's documentation: https://docs.djangoproject.com/en/2.0/ref/contrib/admin/