5.2. The Default User Model

Before creating a custom user model, let's take a look at the default user model included with `django.contrib.auth`, which is included in the `INSTALLED_APPS` setting of new projects:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

1. With `djangojokes.com` open in the terminal, run the following to open the shell:

```
(.venv) …/projects/djangojokes.com> python manage.py shell
```

39. Well, you could, but in this case, it's not worth the trouble.

2. Import the default user model and inspect it:

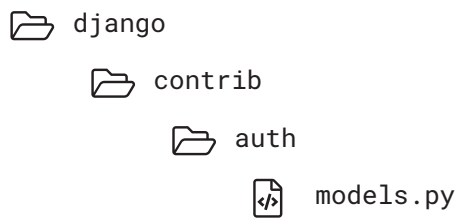
```
>>> from django.contrib.auth.models import User
>>> User
<class 'django.contrib.auth.models.User'>
```

   This indicates that the `User` class is defined in the `django/contrib/auth/models.py` file.

3. Exit out of the Python shell:

```
>>> exit()
```

Find the file that contains the `User` class definition. Starting from `.venv/lib/site-packages/django`, look in:



Let's see how the class is defined:

1. Open `models.py` in your editor and search the file for "class User(". You should find this:

```
class User(AbstractUser):
    …
```

   That indicates that `User` inherits from `AbstractUser`.

2. Search the file for "class AbstractUser". You should find a class that looks something like this:

Demo 5.1: AbstractUser

```
1.    class AbstractUser(AbstractBaseUser, PermissionsMixin):
2.        """
3.        An abstract base class implementing a fully featured User model with
4.        admin-compliant permissions.
5.
6.        Username and password are required. Other fields are optional.
7.        """
8.        username_validator = UnicodeUsernameValidator()
9.
10.       username = models.CharField(…)
11.       first_name = models.CharField(_('first name'), max_length=30, blank=True)
12.       last_name = models.CharField(_('last name'), max_length=150, blank=True)
13.       email = models.EmailField(_('email address'), blank=True)
14.       is_staff = models.BooleanField(…)
15.       is_active = models.BooleanField(…)
16.       date_joined = models.DateTimeField(_('date joined'), default=timezone.now)
```

Notice that the `AbstractUser` class contains the following fields:

A. `username`

B. `first_name`

C. `last_name`

D. `email`

E. `is_staff`

F. `is_active`

G. `date_joined`

`AbstractUser` gets additional fields from the two classes it inherits from:

AbstractBaseUser

A. `password`

B. `last_login`

PermissionsMixin

A. `is_superuser`

While the default user model that comes baked in to Django may be fine for your initial plans, there will likely come a time when you will want to make some modifications to it. For example, you might want to add a `date_of_birth` or a `timezone` field. As the `User` model is part of the built-in

`django.contrib.auth` app, you won't be able to make changes to the model.[40] As such, you should *always* create a custom user model when starting a new project. This will give you full control over the user model as your project develops. And, as it turns out, it's easy to do.

40. Well, you could, but you shouldn't.

Exercise 23: Creating a Custom User Model

20 to 30 minutes

In this exercise, you will create a new app with a custom user model in the `djangojokes` project.

1. With `djangojokes.com` open in the terminal, run the following command to create the **users** app:

```
(.venv) …/projects/djangojokes.com> python manage.py startapp users
```

2. Open `users/models.py` for editing, and add the following code:

Exercise Code 23.1: users/models.py

```
1.  from django.contrib.auth.models import AbstractUser
2.  from django.db import models
3.
4.  class CustomUser(AbstractUser):
5.      pass
```

Things to notice:

- You import `AbstractUser` from `django.contrib.auth.models`.
- The `CustomUser` class inherits from `AbstractUser`, but doesn't change it at all. While you haven't made any changes yet, creating this subclass of `AbstractUser` gives you the option of making changes in the future.

3. Open `djangojokes/settings.py` for editing:

A. Add the new **users** app to the `INSTALLED_APPS`:

```
INSTALLED_APPS = [
    …

    # Local apps
    'common.apps.CommonConfig',
    'jokes.apps.JokesConfig',
    'jokes.apps.PagesConfig',
    'users.apps.UsersConfig',
]
```

B. Immediately below the `AUTH_PASSWORD_VALIDATORS` setting,[41] set `AUTH_USER_MODEL` to `users.CustomUser`:

```
# AUTHENTICATION SETTINGS
AUTH_USER_MODEL = 'users.CustomUser'
```

The `AUTH_USER_MODEL` setting[42] sets the model used to represent a `User` in the project. It defaults to `'auth.User'`. You are overriding that default to set it to the `CustomUser` class you just created.

❖ E23.1. Migrating

Remember that you waited to migrate (see page 171) until you created the custom user model? That is because the initial migration will create the user model. Once that user model is created, you have lost your chance to customize it. So, always *always* **always**, create the custom user model before running the initial migration.

Wait, when do you create the custom user model?

Great question! If you are going to create a custom user model, which you almost definitely should, you must create it *before doing the initial migration*. Always.

41. It could go anywhere in the settings file, but this is a good place for it.
42. https://docs.djangoproject.com/en/3.1/ref/settings/#auth-user-model

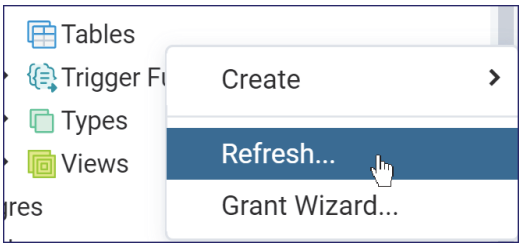
But wait again! Didn't I already run the migrations?

Yes, you did run migrations for `djangojokes` already (see page 62). But since you replaced the SQLite database with PostgreSQL, you need to run them again, which gives you a fresh start.

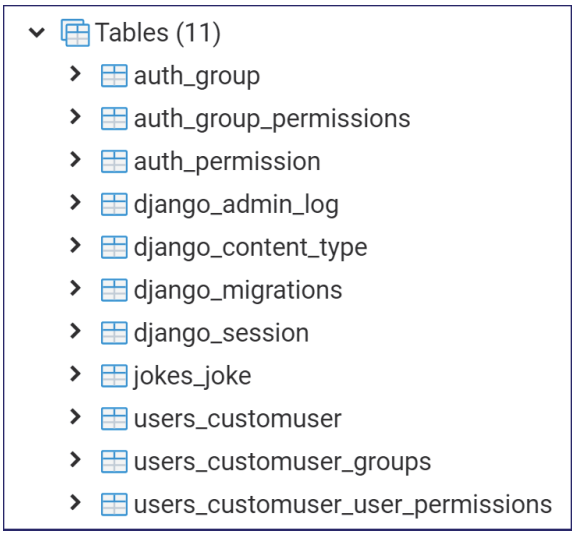
1. Run `makemigrations` and `migrate` to make the new migration files for the `CustomUser` model and run all the project's migration files:

```
(.venv) …/projects/djangojokes.com> python manage.py makemigrations
Migrations for 'users':
  users\migrations\0001_initial.py
    - Create model CustomUser
(.venv) …/projects/djangojokes.com> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, jokes, sessions, users
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0001_initial... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying users.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying jokes.0001_initial... OK
  Applying sessions.0001_initial... OK
```

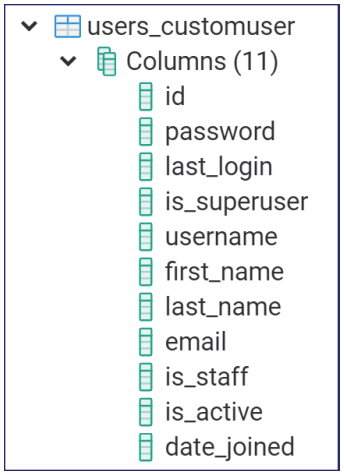
2. Open pgAdmin and navigate to **Databases > jokes > Schemas > public > Tables**. Right-click **Tables** and select **Refresh**:



You should see the following tables:



3. Expand **users_customuser** and **Columns**:



Notice that it contains the same fields as the `AbstractUser` class you looked at earlier (see page 173). In addition, it includes an `id` field. By default, every table created in Django will get an `id` primary key field that is an auto-incrementing integer.

Git Commit

Commit your code to Git.

5.3. Referencing the User Model

There are two commonly used ways of referencing the user model:

1. The `get_user_model()` method from the `django.contrib.auth` app.
2. The `AUTH_USER_MODEL` variable in the project `settings`.

You should use `AUTH_USER_MODEL` when referencing the model as a foreign key to other models. You will see this when you associate users with the jokes they create (see page 343).

In most other cases, you should use `get_user_model()`.