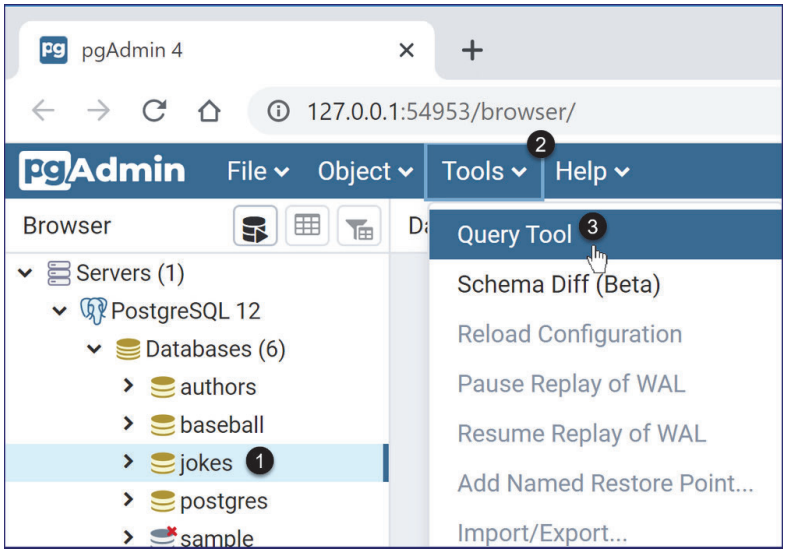
Exercise 24: Getting Started with Django Admin

25 to 40 minutes

Most database-driven websites include an administrative site for managing users and other data. Often, these sites simply provide CRUD (**C**reate, **R**ead, **U**pdate, **D**elete) functionality. Django projects include a built-in administrative site known as *Django admin*.

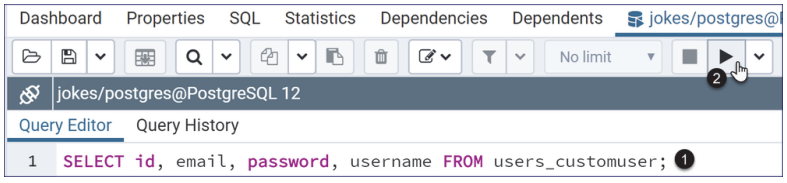
As you have seen, new Django projects have a default user model baked in. Only superusers (`is_superuser == True`) and staff (`is_staff == True`) have access to Django admin. After creating a user model and making any necessary migrations, the first thing you usually do is create a superuser. But before you do that, let's demonstrate that there are currently no users in the database:

1. In pgAdmin, with the **jokes** database selected on the left, click **Tools > Query Tool**:

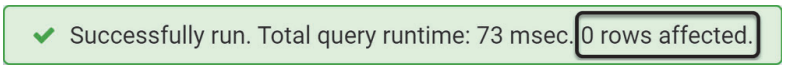


2. Enter the following query and click the black triangle to run it:

```
SELECT id, email, password, username FROM users_customuser;
```



In the bottom right of pgAdmin, you'll get a message indicating that there are no users:



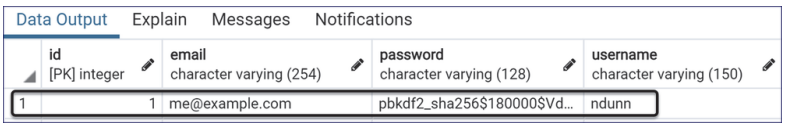
Now, go ahead and create a superuser:

1. In the terminal, run:

```
(.venv) …/projects/djangojokes.com> python manage.py createsuperuser
Username: ndunn
Email address: me@example.com
Password:
Password (again):
Superuser created successfully
```

   A. **Username**: Enter whatever username you like or accept the default.
   B. **Email address**: Use an email address at which you can receive emails.
   C. **Password**: Enter a secure password (and remember it).
   D. **Password (again)**: Repeat the password.

Now, go back to pgAdmin and run the same SQL query again. This time, you will get a result:



pgAdmin in Django Development

In normal Django development, you do not need to use pgAdmin to inspect the database, but it can be helpful from time to time, especially when you are learning Django.

❖ E24.1. Registering the Custom User

You need to register your custom user with Django admin. You do this in the `users/admin.py` file. Open that file up and add the following content:[43]

43. **Don't want to type?** Copy from `starter-code/django-admin/users_admin.py`.

Exercise Code 24.1: users/admin.py

```
1.  from django.contrib import admin
2.  from django.contrib.auth import get_user_model
3.  from django.contrib.auth.admin import UserAdmin
4.
5.  CustomUser = get_user_model()
6.
7.  @admin.register(CustomUser)
8.  class CustomUserAdmin(UserAdmin):
9.      model = CustomUser
10.
11.     add_fieldsets = UserAdmin.add_fieldsets + (
12.         ('Optional Fields', {
13.             'classes': ('wide',),
14.             'fields': ('email', 'first_name', 'last_name'),
15.         }),
16.     )
```

Things to notice:

1. On the first line, you import `admin` from `django.contrib`. This is one of the apps included in `INSTALLED_APPS`.
2. You import `get_user_model()` from `django.contrib.auth`:

   ```
   from django.contrib.auth import get_user_model
   ```

3. As you have overridden the default user model, you need to override the default `UserAdmin` class as well:
   A. You import `django.contrib.auth.admin`'s `UserAdmin` class, which is the class used for managing the default user model:

      ```
      from django.contrib.auth.admin import UserAdmin
      ```

   B. You create a `CustomUserAdmin` class by inheriting from `UserAdmin` and set `model` to `CustomUser`:

      ```
      class CustomUserAdmin(UserAdmin):
      model = CustomUser
      ```

4. You append to the `add_fieldsets` attribute of the `UserAdmin` class. More on this in a moment.
5. You must register `CustomUserAdmin`. You do this using the `@admin.register` decorator:

```
@admin.register(CustomUser)
```

Another way of registering a `ModelAdmin` class is to use the `register()` method:

```
admin.site.register(CustomUser, CustomUserAdmin)
```

But the decorator is more convenient.

`fieldsets` and `add_fieldsets`

The `fieldsets` and `add_fieldsets` properties hold the fieldsets that show up in Django admin's forms:

- `fieldsets` – These are the fieldsets used for updating existing records. Generally, you want this to contain all the fields that can be modified.
- `add_fieldsets` – These are the fieldsets used for creating new records. This only needs to contain the required fields; however, it can contain additional fields that you generally want to have data for in new records.

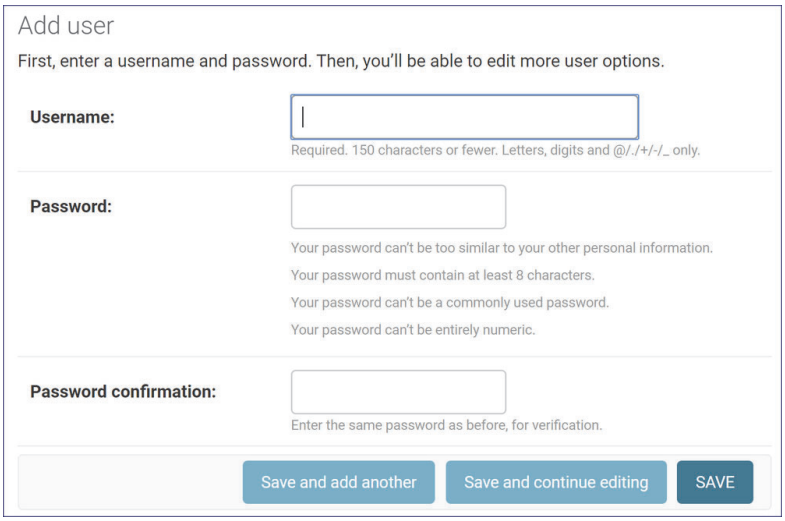
`fieldsets` and `add_fieldsets` each holds a tuple, which itself contains one or more 2-element tuples:

1. The first element is the fieldset name (or `None` if the fieldset is unnamed).
2. The second element is a dictionary containing the CSS classes and the fields in the fieldset.

The `add_fieldsets` attribute of the built-in `UserAdmin` class looks like this:

```
add_fieldsets = (
    (None, {
        'classes': ('wide',),
        'fields': ('username', 'password1', 'password2'),
    }),
)
```

It contains only one tuple, meaning there is only one fieldset. The `None` value indicates that the fieldset has no name. It has one class: "wide" and three fields: "username," "password1," and "password2". The resulting **Add User** form looks like this:



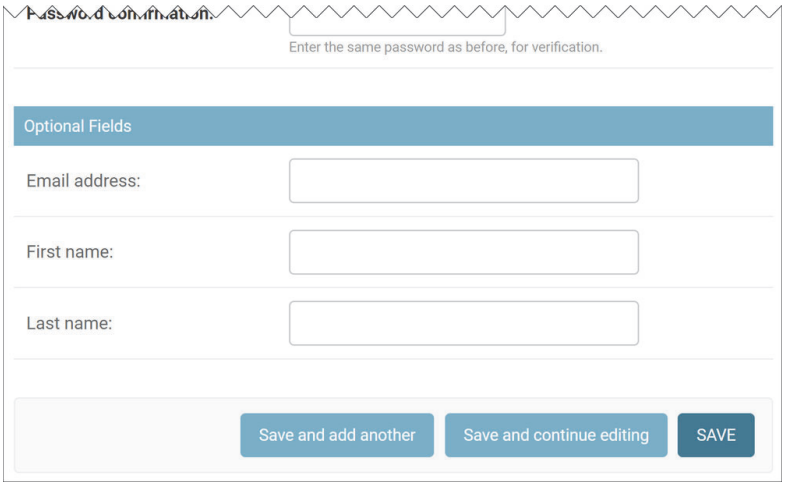
In your `CustomUserAdmin` class, you append[44] another tuple to `UserAdmin`'s `add_fieldsets`:

```
add_fieldsets = UserAdmin.add_fieldsets + (
    ('Optional Fields', {
        'classes': ('wide',),
        'fields': ('email', 'first_name', 'last_name'),
    }),
)
```

You've named the new fieldset "Optional Fields," which also will have the "wide" class, and will have three additional fields: "email," "first_name," and "last_name."

These fields will now be added to Django admin's **Add user** form:

44. You cannot really **append** to a tuple, as tuples are immutable. What you are actually doing is overwriting the `add_fieldsets` property with a new value: its previous value plus the new tuple.



Notice the "Optional Fields" header. That comes from this:

```
add_fieldsets = UserAdmin.add_fieldsets + (
    ('Optional Fields', …
)
```

You will be able to use this same technique to customize other forms in Django admin as you add models to your project.

❖ E24.2. Django admin

It is time to check out Django admin:

1. Open `djangojokes/urls.py` in your editor. Notice the first URL pattern:

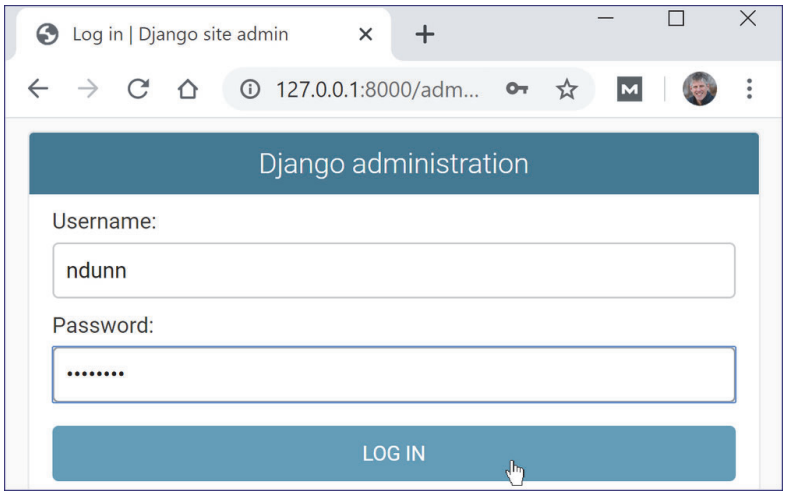
```
urlpatterns = [
    path('admin/', admin.site.urls),
    …
]
```

   This shows the path to Django admin is `admin/`.

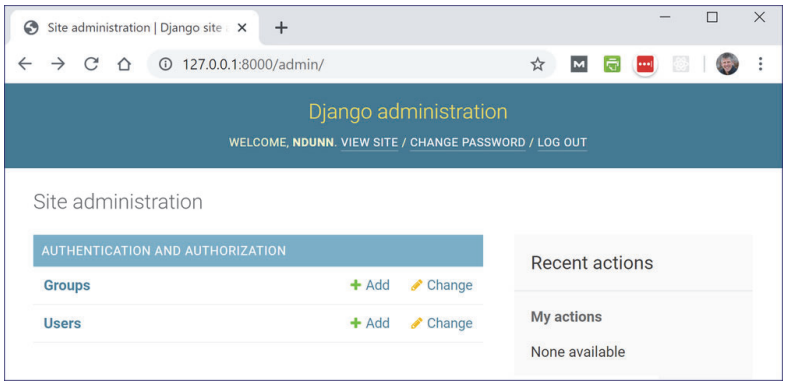
2. Start up the server:

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

3. Point your browser to `http://127.0.0.1:8000/admin/`. You should see a login screen. Enter the username and password you used when creating the superuser:

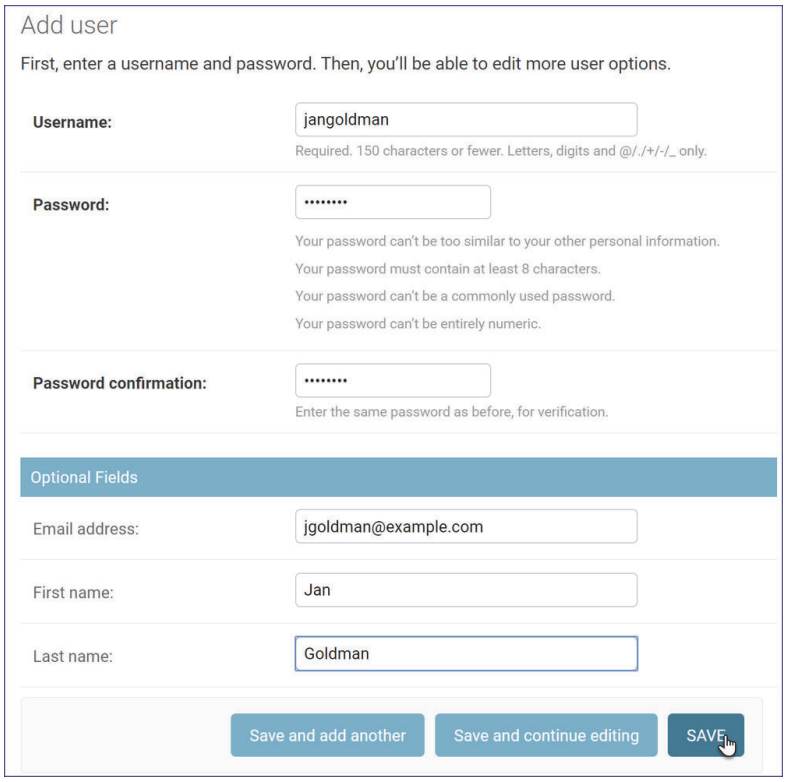


4. You should see a page like this one:



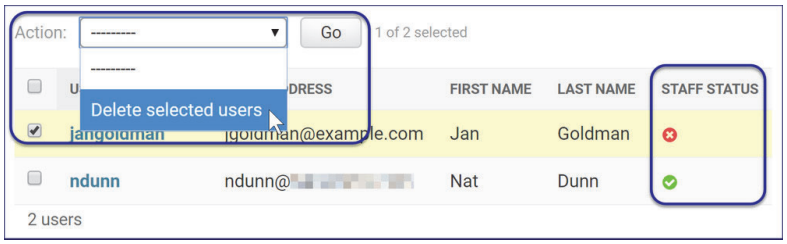
You will see two models: **Groups**[45] and **Users**.

5. Click **Users** and then click your username.
6. Enter your first and last names. Then, scroll down to the bottom and click **Save**.
7. Notice the **Add User** button in the upper right. Click that, fill out the form with a new user, and click **Save**:



8. Click **Users** to go back to the list of users and notice the following:

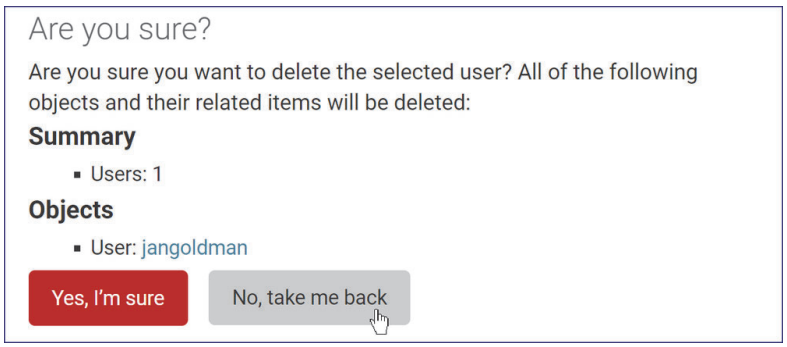
45. The **Groups** model allows you to categorize users into generic groups. This is particularly useful for applying permissions.



A. You are marked as staff, but the new user is not.

B. The **Action** menu allows you to delete selected users.

9. With the new user checked, select **Delete selected users** from the **Action** menu, and click the **Go** button. This will take you to confirmation screen:



Click **No, take me back**.

What we have shown here is known as *CRUD* (for **C**reate, **R**ead, **U**pdate, **D**elete) functionality.

Git Commit

Commit your code to Git.