Theory: Registration and authentication

24 minutes 0 / 5 problems solved

To access service's customization, users should have personal accounts so that they can perform actions related only to them: for example, save personal preferences, post articles, or make purchases.

If you are an active Internet user, you're probably familiar with the sign-up, login and logout operations. Django provides the means to add those operations easily. In this topic, you will learn how to add registration and authentication to a service.

§1. User Model

To distinguish one user from another, we need to store their identification information on the server. On websites, it's usually a unique username or email address. Both characteristics may be stored in the default `User` model.

> To start working with the User model, run `python manage.py migrate` from the root of your project. If you don't change any settings, you will have an SQLite database attached to your project.

After you've made initial migrations you can create new accounts. Mostly, the clients of your service will create accounts by themselves, but let's see how you can create a *usual user* and a *superuser* in the database from the console.

Superuser is an admin account for your service. Being a superuser you can access and manipulate any data in the database. Usual users, by contrast, can manipulate only their own data.

To use an initialized console client, run `python manage.py shell` command.

```
from django.contrib.auth.models import User


User.objects.create_superuser(
   username='admin', email='admin@example.com', password='SeCreTPaSsWorD'
)

User.objects.create_user(
username='usual_user', email='user@example.com', password='NotSecRetAtAll'
)
```

As you may have guessed, `create_user` method creates a usual user and `create_superuser` creates a superuser in your database. You're not likely to need the `create_user` method often, but you will need a console method to make the first admin account.

> Another way to create a superuser is the helper command using with manage.py module.

§2. Preparing URLs

To separate each action, we should choose the URL addresses for login, logout, and signup operations. You should update the `urlpatterns` variable in your main *urls.py* module.

To make it neat and clear, we choose straightforward paths:

Current topic:

Registration and authentication

Topic depends on:

- ✕ Django ORM
- ✕ Forms and validation

Topic is required for:

- Admin interface

Table of contents:

```
urlpatterns += [
    path('login', MyLoginView.as_view()),
    path('logout', LogoutView.as_view()),
    path('signup', MySignupView.as_view()),
]
```

Now we need to implement several classes and import them to the *urls.py* module. We start by making `MySignupView` class.

§3. Signup

You already know how you can create a new user with Python, but regular users don't know Python. We've got to provide a simple web interface for them with an HTML form. Fortunately, making it will only take us a few simple steps, and then our new users will just sign up on their own.

```
from django.contrib.auth.forms import UserCreationForm
from django.views.generic import CreateView


class MySignupView(CreateView):
    form_class = UserCreationForm
    success_url = 'login'
    template_name = 'signup.html'
```

To be able to create objects with the HTTP handler, we inherit `MySignupView` class from `CreateView`. We define several attributes that will do the work for us:

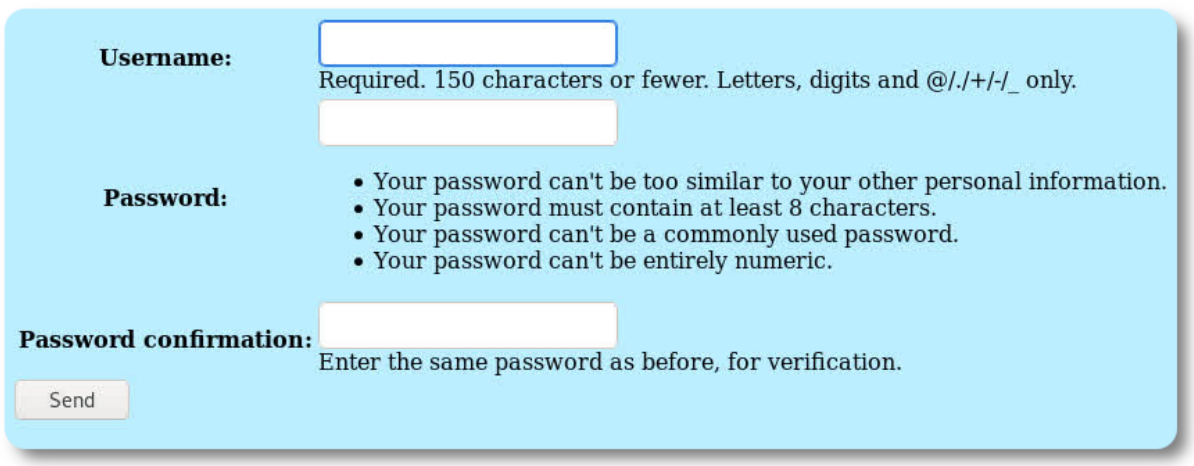
- The `form_class` attribute is a Django form class. We select `UserCreationForm` from the framework to create a new user.
- After our users finished registration, they are redirected to the `success_url` page of the service, in our case, to the login page.
- `template_name` is simply the name of a template responsible for the signup page of the service.

We're almost there with preparing our registration form; just add a custom *signup.html* template and that's it.

```
<form method="post" action="signup">{% csrf_token %}
  <table>{{ form.as_table }}</table>
  <button type="submit">Send</button>
</form>
```

Do not forget to update `settings.TEMPLATES.DIRS` and add the *signup.html* template's directory to it.

We make a simple form and add it to the template, which is enough for a quick start. The page is cluttered with hints, but it does the job well. Now we have a registration form for the service:



§4. Login

The process of creating a request handler for logging in is very similar to making a registration form. We define the class and specify the `template_name` in it:

```
from django.contrib.auth.views import LoginView


class MyLoginView(LoginView):
    redirect_authenticated_user = True
    template_name = 'login.html'
```

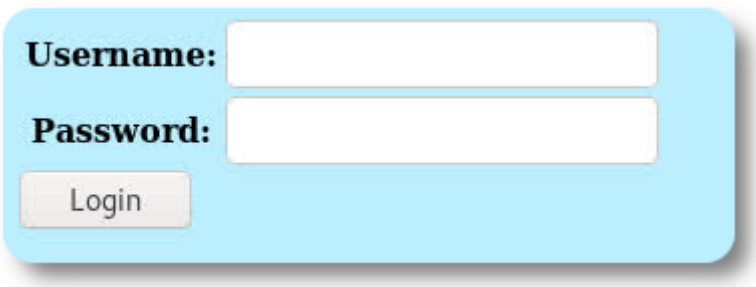
This time we add another attribute `redirect_authenticated_user` and set it to `True`. All authenticated users that come back to the *login* page will be redirected to the main site instead of having to fill the authentication form again.

To define where the user should be redirected after successful authentication, we set `LOGIN_REDIRECT_URL = '/'` in the *settings.py* module. It's usually the main page of the service, but you can choose any page you like.

The *login.html* template differs from *signup.html* by the `action` field and the label of the button:

```
<form method="post" action="login">{% csrf_token %}
  <table>{{ form.as_table }}</table>
  <button type="submit">Login</button>
</form>
```

The result is concise and pleasing:



§5. Logout

The last action our users need is logout. They do not need to send any information, so to log out they should just go to the right URL.

You can see that for login and signup we define our custom `MySignupView` and `MyLoginView` classes accordingly. In `urlpatterns` you can find that for logout we use `LogoutView` class from *django.contrib.auth.views* module. Just import it to the *urls.py* module to complete the work.

If you want to specify where the user should be redirected after logging out, you can define this in the *settings.py* module. For example, to redirect users back to the login page, add this line to the module: `LOGOUT_REDIRECT_URL = '/login'`.

Now you know enough to add authentication to any service, so you can get your hands on personalizing your web service for each individual user.

Report a typo

34 users liked this theory. **3** didn't like it. **What about you?**

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion