Custom User Model, and forms.py in Django

Chapter 7

1

Additional User Attributes

- In the previous chapter, we have used the default User Model provided by Django.
- What if all the provided attributes that the User model provides isn't enough?
- Say, we want to include an additional attribute for each user account, e.g. an age field.
- This is achieved through the creation of an additional model in models.py file and updating settings.py to tell Django to use the new custom user model in place of the built-in User model.

2

Creating our custom user model

Creating our custom user model requires the following steps:

1. Create a new CustomUser model (models.py);
2. Update settings.py to tell Django to use the new custom user model in place of the built-in User model;
3. Create new forms for UserCreation and UserChangeForm (forms.py);
4. Update views.py to use the new forms created in step 3;
5. Update admin.py to use the new CustomUser model and the two new forms created in step 3;
6. Create a migration record for the model and migrate the change into our database to create a new database that uses the custom user model.

3

Creating our custom user model – Step 1: Create a new CustomUser model

Edit models.py to create a database model called User.

- We added our first extra field for the "age" of our users.
- We used Django's PositiveIntegerField which means the integer must be either positive or zero.
- We extend AbstractUser, so our CustomUser is basically a copy of the default User model. The only update is our new age field.

```
from django.contrib.auth.models import AbstractUser
from django.db import models

# Create your models here
class CustomUser (AbstractUser):        # add the model to extend the AbstractUser
    age = models.PositiveIntegerField(default=0)
```

4

Creating our custom user model – Step 2: Update settings.py

Update settings.py to tell Django to use the new custom user model in place of the built-in User model.

- At the bottom of settings.py, add the following line to use CustomUser that we have created in models.py in the previous step:

  AUTH_USER_MODEL = 'yourAppNameHere.CustomUser'

5

Creating our custom user model – forms.py Step 3: Create new forms for UserCreation

In the previous chapter, for (**SignUp – (4)** Write the logic for the view SignUpView), we used Django's built-in form class, UserCreationForm, to build the signup page to register new users easily.

Now, we need to create forms.py to add the forms to interact with our new CustomUser model.

- One case is when a user signs up for a new account on our website.
- The other is within the admin app which allows superusers, to modify existing users.
- So we need to update the two built-in forms for this functionality: UserCreationForm and UserChangeForm.

- For both forms we are setting the model to our CustomUser and using the default fields by using Meta.fields.
- Our CustomUser model contains all the fields of the default User model and our additional age field which we set.

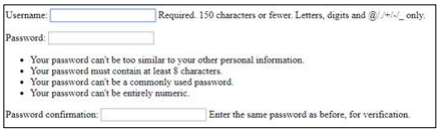
```
from django import forms
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUser

class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = CustomUser
        fields = UserCreationForm.Meta.fields

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = CustomUser
        fields = UserChangeForm.Meta.fields
```

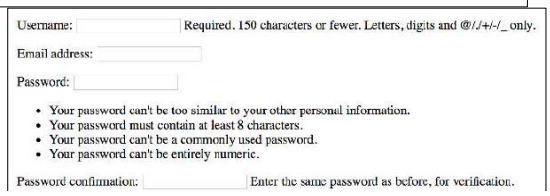
Creating our custom user model – forms.py
Step 3: Create new forms for UserCreation

- Under fields we're using Meta.fields which just displays the default settings of username/password.
- We can explicitly set which fields to be displayed, so let's update it to ask for a username/email/password by setting it to ('username', 'email',). We don't need to include the password field because it's required.

```
from django import forms
from django.contrib.auth.forms import UserCreationForm, UserChangeForm
from .models import CustomUser

class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = CustomUser
        fields = UserCreationForm.Meta.fields

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = CustomUser
        fields = UserChangeForm.Meta.fields
```

```
class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = CustomUser
        fields = ('username', 'email', ) # new

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = CustomUser
        fields = ('username', 'email', ) # new
```





Creating our custom user model –
Step 4: Update views.py

Update views.py to use the new forms created in step 3.

```
from django.contrib.auth.forms import UserCreationForm
from django.urls import reverse_lazy
from django.views import generic
class SignUpView (generic.CreateView):
    form_class = UserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'signup.html'
```

Example of view from SignUp with the built-in User Model

```
from django.urls import reverse_lazy
from django.views import generic

from .forms import CustomUserCreationForm

class SignUp(generic.CreateView):
    form_class = CustomUserCreationForm
    success_url = reverse_lazy('login')
    template_name = 'signup.html'
```

8

Creating our custom user model – Step 5: Update admin.py

Since Admin is tightly coupled to the default User model. We will extend the existing UserAdmin class to use our new CustomUser model and our two new forms created in step 3.

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .forms import CustomUserCreationForm, CustomUserChangeForm

# Register your models here.
from .models import CustomUser

class CustomUserAdmin(UserAdmin):
    add_form = CustomUserCreationForm
    form = CustomUserChangeForm
    list_display = ['email', 'username', 'age']
    model = CustomUser

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

Customizing lists – list_display:
By default, the list displays the result of __str__() for each object.
To add other fields to the list to display, define a UserAdmin class for the model.

- the add_form is used to create a *new* user;
- the form is to *change* the data of an existing one
- list_display is used to customize which fields to be displayed in Django Admin

Next and final step, we create a migration record for the model and migrate the change into our database to create a new database that uses the custom user model.

Step 6: Create a migration record and migrate the change that uses the custom user model

- It is NOT recommend to run migrate on new projects until *after* a custom user model has been configured.
- Otherwise Django will bind the database to the built-in User model which is difficult to modify later on in the project.

Summary

- Customer User Model with **additional User attributes**