User Profiles

You may be wondering, we already have a user model, then why do we need to have a separate model for profile ? Why not put all the fields in user model itself ?

The answer is, its always better for one model to handle only one functionality, otherwise you will see that your models will get bigger by the day and it becomes difficult to maintain them.

User model is best suited for creating and managing a user, but often we may want to register much more data such as age, profile picture, location, description, about me, etc. A **profile** model is perfectly suited for all these functionality.

Creating Profile

Creating **profile** may look a little tedious at first, but no worries, we will break it down in very simple steps, to give you a big picture the overall steps are -

1. Create a profile model and assign it to a user model.
2. Tell Django to first create and then save profile as soon as a new user is registered.
3. Add profile to our admin.

Profile Model

To begin with, let's keep our model simple, later on we can add fields as required.

```
# Profile
class Profile(models.Model):
    # One to one relationship with User
    user = models.OneToOneField(MyUser, on_delete=models.CASCADE)
    # Custom fields
    age = models.PositiveIntegerField(blank=True, null=True)
    # To create and save profile we will use 'signals'
    # check 'signals.py' and 'apps.py' for further reference

    def __str__(self):
        return f"{self.user.username}'s profile"
```

As you can see, we create a **'One To One Field'** with our custom user model, this is to ensure we have only **one** profile per user.

Then we add a field for registering **'age'**, note that it can be left blank while registering a new profile.

Next, is to simply migrate

```
$ python manage.py makemigrations
$ python manage.py migrate
```

We, then add profile to our admin as usual.

Signals

The next question is, how do we tell Django to create a profile and assign it to a user whenever a new user is registered ? We will be using **'signals'** to do exactly this, **'signals'** ship with Django as a built-in module and we simply need to wire it.

Ref - Django Signals

For simplicity, let's imagine that signals have two participating parties, one is the **sender** and other is the **receiver**. Then we need to define who is the sender and what needs to be done by the receiver.

Steps -

1. Create a separate file 'signals.py' in your users app, all the signals code will live inside this file.
2. Hook the signals to our 'apps.py' so that they can be called when ready.

We want to send a signal as soon as a new user is created, so we use 'post_save' signal

Ref - post_save

Ref - Receiver Parameters

```
from django.db.models.signals import post_save
from django.contrib.auth import get_user_model
from django.dispatch import receiver

from .models import Profile

# Get our custom user model
User = get_user_model()

# create a profile when a new user is created
@receiver(post_save, sender=User)
def create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

# save that profile
@receiver(post_save, sender=User)
def save_profile(sender, instance, **kwargs):
    instance.profile.save()
```

Finally, we hook the signals to our app, so that signals can be called as soon as ready.

```
users/apps.py
```

```
from django.apps import AppConfig


class UsersConfig(AppConfig):
    name = 'users'

    # signal is called as soon as registry is populated
    def ready(self):
        import users.signals
```

Signing Up

In the previous section, we created and registered a superuser, but the superuser doesn't have a **profile**, so in order to fix that, we can simply delete our database and migrate again.

Its perfectly normal to destroy and seed database many times during development.

Simply select **'db.sqlite3'** and delete it!

Now, we can migrate our database again -

```
$ python manage.py migrate
```

Create a new superuser -

```
$ python manage.py createsuperuser
```

Login to admin site, now you can see a **profile** assigned to the superuser.