---------- DJANGO TUTORIAL 9 ----------

I start a larger Django project with additional functionality beyond what was covered in our newspaper site. This will be like a craigslist site where users can upload products they want to sell. That means we'll have a more robust user and products model.

In this video I'll show you how to create custom user accounts. We will use the AbstractUser to define these custom user models.

1. Create our project

```
mkdir BobsList
cd BobsList
pipenv install django==3.0
pipenv shell
django-admin startproject bobs_list .
python manage.py startapp users
python manage.py runserver
Check http://127.0.0.1:8000/
```

2. Add our app to settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'users.apps.UsersConfig', # Added users app
]
# Tell Django to use our custom user model
AUTH_USER_MODEL = 'users.CustomUser'
```

3. Define our custom user in users/models.py

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

class CustomUser(AbstractUser):
    # Define we want to store users age
    # State that age can be null or have an empty value
    # Define that age is an integer
    # Other data types include BooleanField, DateField,
    # TimeField, DateTimeField, AutoField (Auto Increments),
    # DecimalField(decimal_places=X, max_digits=X),
    # FloatField, IntegerField, CharField, TextField,
    # EmailField, FileField, ImageField, GenericIPAddressField,
    # URLField
    street = models.CharField(max_length=40)
    city = models.CharField(max_length=50)
    state = models.CharField(max_length=20)
    zip_code = models.PositiveIntegerField(null=True, blank=True)
    phone = models.CharField(max_length=20, null=True, blank=True)
    age = models.PositiveIntegerField(null=True, blank=True)
```

4. Create users/forms.py and add the ability to create and update user accounts

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

# Add creation and change forms and use Meta to add
# additional user data to the default fields
# The default fields we inherit from
# UserCreationForm.Meta.fields include first_name,
# last_name, username, email, password, groups,
# user_permissions, is_staff (access admin),
# is_active (Set to False vs. deleting), is_superuser,
# last_login, date_joined
class CustomUserCreationForm(UserCreationForm):
    class Meta(UserCreationForm):
        model = CustomUser
        # This says to use defaults + additional data
        # This defines the exact fields to use in the form
        fields = ('username', 'email', 'street', 'city', 'state', 'zip_code', 'phone', 'age',)

class CustomUserChangeForm(UserChangeForm):
    class Meta:
        model = CustomUser
        # Use the default fields
        # fields = UserChangeForm.Meta.fields
        # Or, define the exact fields
        fields = ('username', 'email', 'street', 'city', 'state', 'zip_code', 'phone', 'age',)
```

5. Update users/admin.py with our custom user model

```
# Import admin interface
from django.contrib import admin
# Import user admin class
from django.contrib.auth.admin import UserAdmin
# From forms.py import custom user forms
from .forms import CustomUserCreationForm, CustomUserChangeForm
# Import CustomUser from models.py
from .models import CustomUser

class CustomUserAdmin(UserAdmin):
    # Used to create a new user
    add_form = CustomUserCreationForm
    # Used to change user info
    form = CustomUserChangeForm
    # Model used
    model = CustomUser
    # Defines what is displayed about users in admin
    list_display = ['email', 'username', 'street', 'city', 'state', 'zip_code', 'phone', 'age', 'is_staff',]

# Define the options wanted for this object
```

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

6. Activate our model by making a migrations file which tracks all changes and then we build the DB with the migrate command

Ctrl+C to stop server

```
python manage.py makemigrations users
python manage.py migrate
```

7. Create the superuser & run server

```
python manage.py createsuperuser
python manage.py runserver
```

Open http://127.0.0.1:8000/admin

8. Create directories templates & templates/registration in the main project file on the same level as bobs_list and users

9. Tell settings about the templates folder and where to send users after a login and logout

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],

LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'home'
```

10. Create base html file that other templates will extend and put it in the templates folder

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <!-- Extending templates can add a title -->
  <title>{% block title %}Bob's List{% endblock title %}</title>
</head>
<body>
  <main>
    <!-- This is where other templates will put custom info -->
    {% block content %}
    {% endblock content %}
  </main>
</body>
</html>
```

11. Create home.html in the templates directory

```
{% extends 'base.html' %}
{% block title %}Home{% endblock title %}

{% block content %}
<!-- Check if user is logged in and if they are say hi
and provide a logout option -->
{% if user.is_authenticated %}
```

```
  Hi {{ user.username }}
  <p><a href="{% url 'logout' %}">Log Out</a></p>
{% else %}
  <!-- If not logged in display links to login or signup -->
  <a href="{% url 'login' %}">Log In</a>
  <a href="{% url 'signup' %}">Sign Up</a>
{% endif %}
{% endblock content %}
```

12. Create login.html in the templates/registration folder

```
{% extends 'base.html' %}
{% block title %}Login{% endblock title %}
{% block content %}
<h2>Log In</h2>
<form method="post">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Log In</button>
</form>
{% endblock content %}
```

13. Create signup.html in the templates folder

```
{% extends 'base.html' %}
{% block title %}Sign Up{% endblock title %}
{% block content %}
<h2>Sign Up</h2>
<form method="post">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Sign Up</button>
</form>
{% endblock content %}
```

14. Create users/views.py which will place the sign up form in the page as well as the redirect after sign up and the template name to use

```
from django.urls import reverse_lazy
from django.views.generic import CreateView
from .forms import CustomUserCreationForm

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

15. Create users/urls.py

```
from django.urls import path
from .views import SignUpView

urlpatterns = [
path('signup/', SignUpView.as_view(), name='signup'),]
```

16. Define URL route to home in bobs_list/urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.views.generic.base import TemplateView

# You can use the built-in TemplateView as a temporary
# place holder for the homepage
urlpatterns = [
    path('admin/', admin.site.urls),
    path('users/', include('users.urls')),
    path('users/', include('django.contrib.auth.urls')),
    path('', TemplateView.as_view(template_name='home.html'), name='home'),
]
```

1240