الفصل السابع: حسابات المستخدمين

لقد قمنا حتى الآن ببناء تطبيق مدونة يعمل باستخدام نماذج ولكننا نفتقد جزءًا رئيسيًا من معظم تطبيقات الويب: مصادقة المستخدم. من المعروف أن تنفيذ المصادقة الصحيحة للمستخدم أمر صعب ؛ هناك العديد من المشاكل الأمنية على طول الطريق ، لذا فأنت لا تريد حقًا تنفيذ ذلك بنفسك. لحسن الحظ ، يأتي Django مزودًا بنظام مصادقة مستخدم قوي ومضمّن 106 يمكننا استخدامه وتخصيصه حسب الحاجة. عندما تنشئ مشروعًا جديدًا ، يقوم Django افتراضيًا بتثبيت تطبيق المصادقة ، والذي يوفر لنا كائن مستخدم 107 يحتوي على:

• username • password • email • first_name • last_name

سنستخدم كائن المستخدم هذا لتنفيذ تسجيل الدخول وتسجيل الخروج والتسجيل في تطبيق المدونة الخاص بنا.

Log In

يوفر لنا Django طريقة عرض افتراضية لصفحة تسجيل الدخول عبر LoginView. كل ما نحتاج إلى إضافته هو نمط URL لنظام المصادقة ، ونموذج تسجيل الدخول ، وتحديث صغير لملف config / settings.py الخاص بنا. أولاً ، قم بتحديث ملف config / urls.py. سنضع صفحات تسجيل الدخول والخروج الخاصة بنا على accounts/ URL. هذه إضافة من سطر واحد في السطر التالي إلى الأخير

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
        path('admin/', admin.site.urls),
        path('accounts/', include('django.contrib.auth.urls')), # new
         path('', include('blog.urls')),  ]
```

كما تلاحظ توثيق LoginView ، سيبحث Django افتراضيًا داخل دليل قوالب يسمى registration لملف يسمى login.html لنموذج تسجيل الدخول. لذلك نحن بحاجة إلى إنشاء دليل جديد يسمى registration والملف المطلوب بداخله. من سطر الأوامر ، اكتب Control + c لإنهاء خادمنا المحلي. ثم أدخل ما يلي:

Command Line

```
(blog) $ mkdir templates/registration
(blog) $ touch templates/registration/login.html
```

اكتب الآن كود القالب التالي لملفنا الذي تم إنشاؤه حديثًا

Code:

```
<!-- templates/registration/login.html –>
——————————————————————————
{% extends 'base.html' %}
{% block content %}
<h2>Log In</h2>
<form method="post">
{% csrf_token %}
{{ form.as_p }}
<button type="submit">Log In</button>
</form>
{% endblock content %}
```

نحن نستخدم علامات HTML وتحديد طريقة POST نظرًا لأننا نرسل البيانات إلى الخادم (سنستخدم GET إذا كنا نطلب البيانات ، مثل نموذج محرك البحث). نضيف {٪ csrf_token٪} للمخاوف

الأمنية ، وتحديدًا لمنع هجوم XSS. يتم إخراج محتويات النموذج بين علامات الفقرة بفضل {{form.as_p}} ثم نضيف زر "إرسال". الخطوة الأخيرة هي أننا نحتاج إلى تحديد مكان إعادة توجيه المستخدم عند تسجيل دخول ناجح. يمكننا تعيين هذا باستخدام إعداد LOGIN_REDIRECT_URL. أضف ما يلي في الجزء السفلي من ملف config / settings.py:

Code

```
# config/settings.py
LOGIN_REDIRECT_URL = 'home'
```

سيتم الآن إعادة توجيه المستخدم إلى نموذج "الصفحة الرئيسية" وهي صفحتنا الرئيسية. وقد انتهينا بالفعل في هذه المرحلة! إذا بدأت الآن تشغيل خادم Django مرة أخرى باستخدام خادم إدارة python ، وانتقلت إلى صفحة تسجيل الدخول على العنوان http://127.0.0.1:8000/accounts/login/ ، فسترى ما يلي:

Log in page

عند إدخال معلومات تسجيل الدخول لحساب المستخدم المتميز الخاص بنا ، يتم إعادة توجيهنا إلى الصفحة الرئيسية. لاحظ أننا لم نضف أي منطق view أو ننشئ نموذج قاعدة بيانات لأن نظام مصادقة Django قدم لنا كليهما تلقائيًا. شكرا جانغو!

Updated Homepage

لنقم بتحديث نموذج base.html الخاص بنا حتى نعرض رسالة للمستخدمين سواء قاموا بتسجيل الدخول أم لا. يمكننا استخدام سمة is_authenticated لهذا الغرض. في الوقت الحالي ، يمكننا ببساطة وضع هذا الرمز في موضع بارز. في وقت لاحق يمكننا تصميمه بشكل أكثر ملاءمة. قم بتحديث ملف base.html برمز جديد يبدأ أسفل علامة الإغلاق.

Code

```
<!-- templates/base.html -->
{% load static %}
<html>
<head>
<title>Django blog</title>
<link
href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:400"
rel="stylesh\
eet">
<link href="{% static 'css/base.css' %}" rel="stylesheet"s>
</head>
<body>
<div>
<header>
<div class="nav-left">
<h1><a href="{% url 'home' %}">Django blog</a></h1>
</div>
<div class="nav-right">
```

```
<a href="{% url 'post_new' %}">+ New Blog Post</a>
</div>
</header>
{% if user.is_authenticated %}
<p>Hi {{ user.username }}!</p>
{% else %}
<p>You are not logged in.</p>
<a href="{% url 'login' %}">Log In</a>
{% endif %}
{% block content %}
{% endblock content %}
</div>
</body>
</html>
```

إذا تم تسجيل دخول المستخدم ، فنحن نقول له مرحبًا بالاسم ، وإذا لم يكن الأمر كذلك ، فنحن نقدم رابطًا لصفحة تسجيل الدخول المنشأة حديثًا.

Log Out Link

قمنا بإضافة منطق صفحة نموذجية للمستخدمين الذين تم تسجيل خروجهم ولكن ... كيف نقوم بتسجيل الخروج الآن؟ يمكننا الذهاب إلى لوحة المشرف والقيام بذلك يدويًا ، ولكن هناك طريقة أفضل. دعنا نضيف رابط تسجيل الخروج بدلاً من ذلك يعيد التوجيه إلى الصفحة الرئيسية. بفضل نظام Django auth ، فإن تحقيق ذلك سهل للغاية. في ملف base.html الخاص بنا ، أضف رابط {٪'url 'logout ٪} المكون من سطر واحد لتسجيل الخروج أسفل تحية المستخدم مباشرةً.

```
<!-- templates/base.html-->
```

```
...
{% if user.is_authenticated %}
<p>Hi {{ user.username }}!</p>
<p><a href="{% url 'logout' %}">Log out</a></p>
{% else %}
…
```

هذا كل ما نحتاج إلى القيام به حيث يتم توفير العرض الضروري لنا من خلال تطبيق Django auth. نحن بحاجة إلى تحديد مكان إعادة توجيه المستخدم عند تسجيل الخروج بالرغم من ذلك. قم بتحديث config / settings.py لتوفير رابط إعادة توجيه يسمى ، بشكل مناسب ، -_LOGOUT REDIRECT_URL. يمكننا إضافته مباشرة بجوار إعادة توجيه تسجيل الدخول ، لذا يجب أن يبدو الجزء السفلي من الملف كما يلي:

Code

```
# config/settings.py
LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'home' # new
```

إذا قمت بتحديث الصفحة الرئيسية ، فسترى أنها تحتوي الآن على رابط "تسجيل الخروج" للمستخدمين الذين قاموا بتسجيل الدخول.

والنقر عليه يعيدك إلى الصفحة الرئيسية برابط تسجيل الدخول.

انطلق وحاول تسجيل الدخول والخروج عدة مرات باستخدام حساب المستخدم الخاص بك.

Sign Up

نحتاج إلى كتابة وجهة نظرنا الخاصة لصفحة تسجيل لتسجيل مستخدمين جدد ، لكن Django يوفر لنا فئة نموذج ، UserCreationForm ، لتسهيل الأمور. بشكل افتراضي ، يأتي مع ثلاثة حقول: اسم المستخدم وكلمة المرور 1 وكلمة المرور 2. هناك العديد من الطرق لتنظيم الكود وبنية عنوان URL من أجل نظام مصادقة مستخدم قوي. أوقف الخادم المحلي باستخدام Control + c وأنشئ تطبيقًا جديدًا accounts مخصص لصفحة التسجيل الخاصة بنا

Command Line

(blog) $ python manage.py startapp accounts

أضف التطبيق الجديد إلى إعداد INSTALLED_APPS في ملف config / settings.py الخاص بنا.

Code

config/settings.py

INSTALLED_APPS = [

```
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'blog',
'accounts', # new
]
```

بعد ذلك ، أضف مسار URL جديدًا في config / urls.py للإشارة إلى هذا التطبيق الجديد أدناه مباشرةً حيث نقوم بتضمين تطبيق المصادقة المدمج.

Code

```
# config/urls.py
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
path('admin/', admin.site.urls),
path('accounts/', include('django.contrib.auth.urls')),
path('accounts/', include('accounts.urls')), # new
path('', include('blog.urls')),
]
```

ترتيب عناوين url الخاصة بنا مهم هنا لأن Django يقرأ هذا الملف من أعلى إلى أسفل. لذلك عندما نطلب عنوان url / accounts / signup ، فإن Django سيبحث أولاً في المصادقة ، ولن يعثر عليه ، ثم ينتقل إلى تطبيق الحسابات. دعونا نمضي قدمًا وأنشئ ملف حسابات / urls.py الخاص بنا.

Command Line

(blog) $ touch accounts/urls.py

And add the following code:

Code

```
# accounts/urls.py
from django.urls import path
from .views import SignUpView
urlpatterns = [
path('signup/', SignUpView.as_view(), name='signup'),
]
```

نحن نستخدم طريقة عرض لم يتم إنشاؤها بعد تسمى SignupView والتي نعلم بالفعل أنها تعتمد على الفئة نظرًا لأنها مكتوبة بأحرف كبيرة وتحتوي على اللاحقة as_view (). مساره هو signup/ لذا فإن مسار URL العام سيكون accounts/signup/ الآن لطريقة العرض التي تستخدم UserCreationForm المضمنة و generic CreateView.

Code

```
# accounts/views.py
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 = 'registration/signup.html'
```

نحن نصنف العرض العام المستند إلى الفئة CreateView في فئة SignUpView الخاصة بنا.
نحدد استخدام UserCreationForm المدمج والقالب الذي لم يتم إنشاؤه بعد في signup.html.
ونستخدم reverse_lazy لإعادة توجيه المستخدم إلى صفحة تسجيل الدخول عند التسجيل بنجاح.
لماذا استخدام reverse_lazy هنا بدلاً من عكس؟ والسبب هو أنه بالنسبة لجميع طرق العرض العامة المستندة إلى الفئة ، لا يتم تحميل عناوين URL عند استيراد الملف ، لذلك يتعين علينا استخدام الشكل البطيء للعكس لتحميلها لاحقًا عندما تكون متاحة. فلنقم الآن بإضافة signup.html إلى ملف templates/registration/ directory

Command Line

```
(blog) $ touch templates/registration/signup.html
```

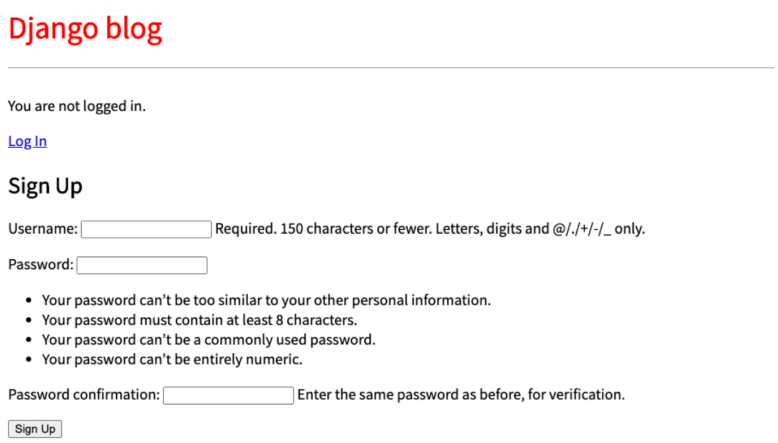
أضف ثم املأها بالرمز أدناه.

Code

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

هذا الشكل مشابه جدًا لما فعلناه من قبل. نقوم بتوسيع القالب الأساسي الخاص بنا في الأعلى ، ضع الكود بين علامات <form></form> ، استخدم csrf_token للأمان ، واعرض محتوى النموذج في علامات الفقرة مع form.as_p ، وقم بتضمين زر إرسال. لقد انتهينا الآن! لاختباره ، ابدأ تشغيل الخادم المحلي باستخدام python manager.py runserver وانتقل إلى

http://127.0.0.1:8000/accounts/signup/



لاحظ أن هناك الكثير من النصوص الإضافية التي يتضمنها Django بشكل افتراضي. يمكننا تخصيص هذا باستخدام شيء مثل إطار عمل الرسائل المدمج 112 ولكن الآن جرب النموذج. لقد أنشأت مستخدمًا جديدًا يسمى "william" وعند الإرسال تمت إعادة توجيهي إلى صفحة تسجيل الدخول. ثم بعد تسجيل الدخول بنجاح باستخدام المستخدم وكلمة المرور الجديدين ، تمت إعادة توجيهي إلى الصفحة الرئيسية من خلال التحية الشخصية "Hi username" الخاصة بنا.

وبالتالي فإن التدفق النهائي لدينا هو: الاشتراك -> تسجيل الدخول -> الصفحة الرئيسية. وبالطبع يمكننا تعديل هذا كما نريد. يقوم SignupView بإعادة التوجيه لتسجيل الدخول لأننا قمنا بتعيين

Success_url = reverse_ lazy (" login").

تقوم صفحة تسجيل الدخول بإعادة التوجيه إلى الصفحة الرئيسية لأنه في ملف config / settings.py الخاص بنا قمنا بتعيين LOGIN_REDIRECT_URL = 'home'.

قد يبدو من الصعب في البداية تتبع جميع الأجزاء المختلفة لمشروع Django. هذا امر عادي. لكني أعدك بمرور الوقت أنهم سيبدأون في جعلهم أكثر منطقية.

Static Files

في السابق ، قمنا بتكوين ملفاتنا الثابتة عن طريق إنشاء مجلد ثابت مخصص ، وتوجيه STATICFILES_DIRS إليه في ملف config / settings.py ، وإضافة {٪ load static ٪} إلى نموذج base.html الخاص بنا. ولكن نظرًا لأن Django لن يقدم ملفات ثابتة في الإنتاج ، فنحن بحاجة إلى بضع خطوات إضافية الآن. التغيير الأول هو استخدام الأمر Collectstatic الخاص بـ Django والذي يجمع كل الملفات الثابتة في جميع أنحاء المشروع في دليل فردي مناسب للنشر. ثانيًا ، يجب علينا تعيين تكوين STATIC_ROOT ، وهو الموقع المطلق لهذه الملفات المجمعة ، إلى مجلد يسمى static files. وثالثًا ، نحتاج إلى تعيين STATICFILES_STORAGE ، وهو محرك تخزين الملفات الذي يستخدمه collectstatic. هذا هو الشكل الذي يجب أن يبدو عليه ملف config / settings.py المحدث:

Code

```
# config/settings.py
STATIC_URL = '/static/'
STATICFILES_DIRS = [str(BASE_DIR.joinpath('static'))]
STATIC_ROOT = STATIC_ROOT = str(BASE_DIR.joinpath('staticfiles'))
# new
STATICFILES_STORAGE =
'django.contrib.staticfiles.storage.StaticFilesStorage' # new
```

Now run the command python manage.py collectstatic:

Command Line

```
(blog) $ python manage.py collectstatic
```

إذا نظرت إلى مجلد مشروعك الآن ، فسترى أنه يوجد مجلد staticfiles جديد يحتوي على مجلدات admin و css. الـ Admin هو الملفات الثابتة المضمنة للمسؤول ، بينما css هو الملف الذي أنشأناه. قبل كل عملية نشر جديدة ، يجب تشغيل الأمر collectstatic لتجميعها في مجلد الملفات الثابتة المستخدمة في الإنتاج. نظرًا لأن هذه خطوة سهلة لنسيانها غالبًا ما يتم تشغيلها تلقائيًا في مشاريع أكبر على الرغم من أن القيام بذلك خارج نطاق مشروعنا الحالي. في حين أن هناك عدة طرق لخدمة هذه الملفات الثابتة المجمعة في الإنتاج ، فإن الطريقة الأكثر شيوعًا - والأسلوب الذي سنستخدمه هنا - هو تقديم حزمة WhiteNoise . للبدء ، قم بتثبيت أحدث إصدار باستخدام Pipenv:

Command Line

```
(blog) $ pipenv install whitenoise==5.1.0
```

ثم في config / settings.py هناك ثلاثة تحديثات يجب إجراؤها:

- إضافة whitenoise إلى INSTALLED_APPS أعلى التطبيق المدمج في الملفات الثابتة
- ضمن MIDDLEWARE أضف سطرًا جديدًا لبرنامج WhiteNoiseMiddleware
- قم بتغيير STATICFILES_STORAGE لاستخدام WhiteNoise

يجب أن يبدو الملف المحدث على النحو التالي:

Code

```
# config/settings.py
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
```

```
'whitenoise.runserver_nostatic', # new
'django.contrib.staticfiles',
'blog',
'accounts',
]
MIDDLEWARE = [
'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'whitenoise.middleware.WhiteNoiseMiddleware', # new
'django.middleware.common.CommonMiddleware',
...
]
STATIC_URL = '/static/'
STATICFILES_DIRS = [str(BASE_DIR.joinpath('static'))]
STATIC_ROOT = str(BASE_DIR.joinpath('staticfiles'))
STATICFILES_STORAGE =
'whitenoise.storage.CompressedManifestStaticFilesStorage' # new
```

نظرًا لتغير أسلوب STATICFILES_STORAGE الخاص بنا ، قم بتشغيل Collectstatic مرة أخرى لاستخدام whitenoise بدلاً من ذلك:

Command Line

```
(blog) $ python manage.py collectstatic
```

سيكون هناك تحذير قصير ، سيؤدي هذا إلى استبدال الملفات الموجودة! هل انت متأكد من أنك تريد أن تفعل هذا؟ اكتب "نعم" واضغط على RETURN. يتم الآن إعادة إنشاء الملفات الثابتة المجمعة في نفس مجلد الملفات الثابتة باستخدام WhiteNoise. تعتبر الملفات الثابتة مربكة جدًا للقادمين الجدد ،

لذا في ما يلي ملخص موجز للخطوات التي نفذناها حتى الآن في موقع المدونة الخاص بنا. أولاً ، بالنسبة للتطوير المحلي في الفصل الخامس ، أنشأنا مجلدًا ثابتًا من المستوى الأعلى وقمنا بتحديث STATICFILES_DIRS للإشارة إليه. في هذا الفصل ، أضفنا تكوينات لـ STATIC_ROOT و STATICFILES_STORAGE قبل تشغيل collectstatic لأول مرة ، والتي جمعت جميع ملفاتنا الثابتة عبر المشروع بأكمله في مجلد واحد ثابت ملفات. أخيرًا ، قمنا بتثبيت whitenoise ، وتحديث INSTALLED_APPS ، و MIDDLEWARE ، و
STATICFILES_STORAGE ، وأعدنا تشغيل Collectstatic. يواجه معظم المطورين ، بمن فيهم أنا ، صعوبة في تذكر كل هذه الخطوات بشكل صحيح والاعتماد على الملاحظات كتذكير ودي

خاتمة

مع الحد الأدنى من التعليمات البرمجية ، قمنا بإضافة تسجيل الدخول وتسجيل الخروج والاشتراك في موقع المدونة الخاص بنا. تحت الغطاء ، اعتنى Django بالعديد من المشاكل الأمنية التي يمكن أن تظهر إذا حاولت إنشاء تدفق مصادقة المستخدم الخاص بك من البداية. لقد قمنا بتكوين الملفات الثابتة بشكل صحيح للإنتاج ونشرنا موقع الويب الخاص بنا ، مرة أخرى ، على Heroku. أحسنت! في الفصل التالي ، سنشرع في المشروع الرئيسي الأخير للكتاب ، وهو موقع صحيفة يستخدم نموذج مستخدم مخصصًا وتدفق تسجيل مستخدم متقدم وتصميمًا محسّنًا عبر Bootstrap وتكوين البريد الإلكتروني. يتضمن أيضًا الأذونات والتراخيص المناسبة ومتغيرات البيئة والمزيد من التحسينات الأمنية لعملية النشر الخاصة بنا