

Contenidos a Trabajar

1. Vistas de autenticación de Django
2. Algunas de las vistas de autenticación más usadas
3. Extendiendo el modelo User existente
 - Sustituyendo un modelo User personalizado

Vistas de autenticación de Django

Django proporciona varias vistas que puede usar para manejar el inicio de sesión, el cierre de sesión y la administración de contraseñas. Estos hacen uso de los formularios de autenticación de stock, pero también puede pasar sus propios formularios.

Django no proporciona una plantilla predeterminada para todas las vistas de autenticación. Debe crear sus propias plantillas para las vistas que desee utilizar. El contexto de la plantilla se documenta en cada vista; consulte todas las vistas de autenticación en la documentación oficial.

Usando las vistas

Existen diferentes métodos para implementar estas vistas en su proyecto. La forma más sencilla es incluir la URLconf proporcionada **django.contrib.auth.urls** en su propia URLconf, por ejemplo:

```
urlpatterns = [  
    path('accounts/', include('django.contrib.auth.urls')),  
]
```

Esto incluirá los siguientes patrones de URL:

```
accounts/login/ [name='login']  
accounts/logout/ [name='logout']  
accounts/password_change/ [name='password_change']  
accounts/password_change/done/ [name='password_change_done']  
accounts/password_reset/ [name='password_reset']  
accounts/password_reset/done/ [name='password_reset_done']  
accounts/reset/<uidb64>/<token>/ [name='password_reset_confirm']  
accounts/reset/done/ [name='password_reset_complete']
```

Las vistas proporcionan un nombre de URL para facilitar la referencia, sobre todo entre las propias vistas. Si desea configurar sus propios nombres de url, tiene que tener cuidado en sobrescribir cada una de las referencias que se poseen en las vistas.

Si desea tener más control sobre sus URL, puede hacer referencia a una vista específica en su URLconf:

```
from django.contrib.auth import views as auth_views
urlpatterns = [
    path('change-password/', auth_views.PasswordChangeView.as_view()),
]
```

Las vistas tienen argumentos opcionales que puede usar para modificar el comportamiento de la vista. Por ejemplo, si desea cambiar el nombre de la plantilla que usa una vista, puede proporcionar el argumento **template_name**. Una forma de hacer esto es proporcionar argumentos de palabras clave en la URLconf, estos se pasarán a la vista. Por ejemplo:

```
urlpatterns = [
    path(
        'change-password/',
        auth_views.PasswordChangeView.as_view(template_name='change-
        password.html'),
    ),
]
```

Todas las vistas están basadas en clases, lo que le permite personalizarlas fácilmente mediante subclases. A esta altura, le recomendamos repasar la documentación sobre vistas basadas en clases en

<https://docs.djangoproject.com/en/3.2/topics/class-based-views/>

Algunas de las vistas de autenticación más usadas

Esta es una lista con algunas de las vistas que **django.contrib.auth** ofrece.

clase LoginView

Nombre de la URL: login

Atributos:

- **template_name:** el nombre de una plantilla que se mostrará para la vista utilizada para iniciar la sesión del usuario. El valor predeterminado es **registration/login.html**.
- **redirect_field_name:** El nombre de un campo **GET** que contiene la URL para redirigir después de iniciar sesión. El valor predeterminado es **next**.
- **authentication_form:** Un invocable (típicamente una clase de formulario) para usar para la autenticación. El valor predeterminado es **AuthenticationForm**.
- **extra_context:** un diccionario de datos de contexto que se agregará a los datos de contexto predeterminados pasados a la plantilla.
- **redirect_authenticated_user:** un valor booleano que controla si los usuarios autenticados que acceden a la página de inicio de sesión serán redirigidos como si acabaran de iniciar sesión correctamente. El valor predeterminado es **False**.
- **success_url_allowed_hosts:** una serie de hosts **set**, además de `request.get_host()`, que son seguros para redirigir después de iniciar sesión. El valor predeterminado es un **set** vacío.

Esto es lo que **LoginView** hace:

- Si se llama a través de **GET**, muestra un formulario de inicio de sesión que publica en la misma URL. Más sobre esto en un momento.
- Si se llama a través de **POST** con las credenciales enviadas por el usuario, intenta iniciar la sesión del usuario. Si el inicio de sesión es exitoso, la vista se redirige a la URL especificada en **next**. Si **next** no se proporciona, se redirige a **settings.LOGIN_REDIRECT_URL** (que por defecto es **/accounts/profile/**). Si el inicio de sesión no es exitoso, vuelve a mostrar el formulario de inicio de sesión.

Es su responsabilidad proporcionar el html para la plantilla de inicio de sesión,

llamada **registration/login.html** de forma predeterminada. A esta plantilla se le pasan cuatro variables de contexto de plantilla:

- **form**: Un objeto **Form** que representa el **AuthenticationForm**.
- **next**: La URL a la que redirigir después de un inicio de sesión exitoso. Esto también puede contener una cadena de consulta.
- **site**: El actual **Site**, según la configuración **SITE_ID**. Si no tiene instalado un framework de sitio, se establecerá en una instancia de **RequestSite**, que deriva el nombre del sitio y el dominio del actual **HttpRequest**.
- **site_name**: un alias para **site.name**. Si no tiene instalado un framework de sitio, se establecerá en el valor de **request.META['SERVER_NAME']**. Para obtener más información sobre los sitios, consulte "El marco de "sitios" en la documentación oficial.

Si prefiere no llamar a la plantilla **registration/login.html**, puede pasar el parámetro **template_name** a través de los argumentos adicionales al método **as_view** en su URLconf. Por ejemplo, esta línea URLconf usaría en su lugar **myapp/login.html**:

```
path('accounts/login/',  
     auth_views.LoginView.as_view(template_name='myapp/login.html')),
```

También puede especificar el nombre del campo **GET** que contiene la URL para redirigir después de iniciar sesión usando **redirect_field_name**. Por defecto, el campo se llama **next**.

Aquí hay una plantilla de muestra **registration/login.html** que puede usar como punto de partida. Se supone que tiene una plantilla **base.html** que define un bloque **content**:

```
{% extends "base.html" %}  
  
{% block content %}  
  
    {% if form.errors %}  
    <p>Your username and password didn't match. Please try again.</p>  
    {% endif %}
```

<codoa codoo/>

```
{% if next %}
    {% if user.is_authenticated %}
        <p>Your account doesn't have access to this page. To proceed, please
        login with an account that has access.</p>
    {% else %}
        <p>Please login to see this page.</p>
    {% endif %}
{% endif %}

<form method="post" action="{% url 'login' %}">
    {% csrf_token %}
    <table>
    <tr>
        <td>{{ form.username.label_tag }}</td>
        <td>{{ form.username }}</td>
    </tr>
    <tr>
        <td>{{ form.password.label_tag }}</td>
        <td>{{ form.password }}</td>
    </tr>
    </table>

    <input type="submit" value="login">
    <input type="hidden" name="next" value="{{ next }}">
</form>

<# Assumes you setup the password_reset view in your URLconf #>
<p><a href="{% url 'password_reset' %}">Lost password?</a></p>

{% endblock %}
```

Agencia de
Aprendizaje
a lo largo
de la vida

clase LogoutView

Cierra la sesión de un usuario.

Nombre de la URL: `logout`

Atributos:

- **next_page:** La URL a la que se redirigirá después de cerrar la sesión. El valor predeterminado es `settings.LOGOUT_REDIRECT_URL`.
- **template_name:** el nombre completo de una plantilla que se mostrará después de cerrar la sesión del usuario. El valor predeterminado es `registration/logged_out.html`.
- **redirect_field_name:** El nombre de un campo **GET** que contiene la URL para redirigir después de cerrar la sesión. El valor predeterminado es `next`. Sobrescribe la URL `next_page` si se pasa el parámetro **GET** dado.
- **extra_context:** un diccionario de datos de contexto que se agregará a los datos de contexto predeterminados pasados a la plantilla.
- **success_url_allowed_hosts:** Un **set** de hosts, además de `request.get_host()`, que son seguros para redirigir después de cerrar la sesión. El valor predeterminado es **set** un vacío.

Contexto de la plantilla:

- **title:** La cadena "Cerrar sesión", localizada.
- **site:** El actual **Site**, según la configuración `SITE_ID`. Si no tiene instalado un framework de sitio, se establecerá en una instancia de **RequestSite**, que deriva el nombre del sitio y el dominio del actual **HttpRequest**.
- **site_name:** un alias para `site.name`. Si no tiene instalado un framework de sitio, se establecerá en el valor de `request.META['SERVER_NAME']`. Para obtener más información sobre los sitios, consulte "El marco de "sitios" en la documentación oficial.

clase PasswordChangeView

Nombre de la URL: `password_change`

Permite a un usuario cambiar su contraseña.

Atributos:

- **template_name:** el nombre completo de una plantilla que se usará para mostrar el formulario de cambio de contraseña. El valor predeterminado es **registration/password_change_form.html** si no se proporciona.
- **success_url:** La URL a la que redirigir después de un cambio de contraseña exitoso. El valor predeterminado es **'password_change_done'**.
- **form_class:** un formulario personalizado de "cambio de contraseña" que debe aceptar un argumento **user** de palabra clave. El formulario es responsable de cambiar realmente la contraseña del usuario. El valor predeterminado es **PasswordChangeForm**.
- **extra_context:** un diccionario de datos de contexto que se agregará a los datos de contexto predeterminados pasados a la plantilla.

Contexto de la plantilla:

- **form:** El formulario de cambio de contraseña (ver **form_class** arriba).

Para más información sobre el resto de las vistas y sus distintas configuraciones, revise la documentación oficial en el siguiente link:

<https://docs.djangoproject.com/en/3.2/topics/auth/default/#module-django.contrib.auth.views>

Extendiendo el modelo User existente

Hay dos formas de ampliar el modelo **User** predeterminado sin sustituir su propio modelo. Si los cambios que necesita son puramente de comportamiento y no requieren ningún cambio en lo que está almacenado en la base de datos, puede crear un modelo de proxy basado en **User**. Esto permite cualquiera de las funciones que ofrecen los modelos de proxy, incluidos los pedidos predeterminados, los administradores personalizados o los métodos de modelos personalizados.

Si desea almacenar información relacionada con **User**, puede usar un modelo **OneToOneField** que contenga los campos para obtener información adicional. Este modelo uno a uno a menudo se denomina modelo de perfil, ya que podría almacenar información no relacionada con la autenticación sobre un usuario del sitio. Por ejemplo, puede crear un modelo de empleado

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

class Employee(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    department = models.CharField(max_length=100)
```

Suponiendo que un empleado Fred Smith existente que tiene un modelo de usuario y de empleado, puede acceder a la información relacionada utilizando las convenciones de modelos relacionados estándar de Django:

```
>>> u = User.objects.get(username='fsmith')
>>> freds_department = u.employee.department
```

Para agregar los campos de un modelo de perfil a la página de usuario en el administrador, defina un **InlineModelAdmin** (para este ejemplo, usaremos un **StackedInline**) en su aplicación **admin.py** y agréguelo a una clase **UserAdmin** que esté registrada con la clase **User**:

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin as BaseUserAdmin
from django.contrib.auth.models import User

from my_user_profile_app.models import Employee
```

```
# Define an inline admin descriptor for Employee model  
# which acts a bit like a singleton
```

```
class EmployeeInline(admin.StackedInline):  
    model = Employee  
    can_delete = False  
    verbose_name_plural = 'employee'
```

```
# Define a new User admin
```

```
class UserAdmin(BaseUserAdmin):  
    inlines = (EmployeeInline,)
```

```
# Re-register UserAdmin
```

```
admin.site.unregister(User)  
admin.site.register(User, UserAdmin)
```

Estos modelos de perfil no son especiales de ninguna manera, son solo modelos de Django que tienen un enlace uno a uno con un modelo de usuario. Como tales, no se crean automáticamente cuando se crea un usuario, pero [django.db.models.signals.post_save](#) se pueden usar para crear o actualizar modelos relacionados según corresponda.

El uso de modelos relacionados da como resultado consultas o uniones adicionales para recuperar los datos relacionados. Dependiendo de sus necesidades, un modelo de usuario personalizado que incluya los campos relacionados puede ser su mejor opción; sin embargo, las relaciones existentes con el modelo de usuario predeterminado dentro de las aplicaciones de su proyecto pueden justificar la carga adicional de la base de datos.

Sustituyendo un modelo User personalizado

Algunos tipos de proyectos pueden tener requisitos de autenticación para los cuales el modelo **User** incorporado de Django no siempre es apropiado. Por ejemplo, en algunos sitios tiene más sentido usar una dirección de correo electrónico como su token de identificación en lugar de un nombre de usuario.

Django le permite sobrescribir el modelo de usuario predeterminado proporcionando un valor para la configuración **AUTH_USER_MODEL** que hace referencia a un modelo personalizado:



```
AUTH_USER_MODEL = 'myapp.MyUser'
```

Este par punteado describe el nombre de la aplicación Django (que debe estar en su **INSTALLED_APPS**) y el nombre del modelo Django que desea usar como modelo de usuario.

Para más información sobre sobrescritura de modelos de usuario y personalización, por favor diríjase a la documentación oficial en:

<https://docs.djangoproject.com/en/4.1/topics/auth/customizing/#extending-the-existing-user-model>