User Authentication Basics

This concept page introduces the basics of implementing user authentication in Django applications. It covers the built-in authentication system, user registration, login and logout functionalities, user permissions and groups, and various authentication-related components provided by Django.

Concept Overview

Topics

* Django’s Built-in Authentication System
* User Registration
* User Login and Logout
* Password Management
* Authentication Views and URLs

Learning Objectives

* Understand the purpose and components of Django’s authentication system
* Learn how to register new users and create user accounts
* Implement user login and logout functionalities
* Manage user passwords securely
* Utilize Django’s built-in authentication views and URLs

Django’s Built-in Authentication System

Django comes with a built-in authentication system that provides a set of models, views, and utilities for handling user authentication. Here’s a breakdown of the core components:

1. **User Model**

The User model serves as the foundation for representing a user within the authentication system. It stores essential user information such as username, password (hashed for security), email address, and other relevant user-related data.

from django.contrib.auth.models import User

# Create a new user

user = User.objects.create\_user('john', 'john@example.com', 'password123')

# Retrieve a user based on username

user = User.objects.get(username='john')

1. **Authentication Middleware**

Django incorporates authentication middleware that seamlessly associates users with incoming requests and grants access to the authenticated user within views and templates.

1. **Authentication Backends**

Authentication backends handle the process of verifying user credentials. Django provides several built-in authentication backends, with the most common being ModelBackend for authentication against the default User model.

User Registration

User registration is the process of creating new user accounts in your application. Django provides the UserCreationFormform and the CreateViewclass-based view to handle user registration.

from django.contrib.auth.forms import UserCreationForm

from django.urls import reverse\_lazy

from django.views.generic import CreateView

class SignUpView(CreateView):

form\_class = UserCreationForm

success\_url = reverse\_lazy('login')

template\_name = 'registration/signup.html'

In this example, the SignUpView uses the UserCreationForm to handle user registration. When a new user is registered, they are redirected to the login page using the success\_url attribute.

User Login and Logout

Django’s authentication system provides built-in views and utilities for handling user login and logout processes.

User Login

from django.contrib.auth.views import LoginView

from django.urls import path

urlpatterns = [

path('login/', LoginView.as\_view(template\_name='registration/login.html'), name='login'),

]

In this example, the LoginView class-based view is used to handle user login. The template\_name attribute specifies the template to be rendered for the login form.

User Logout

from django.contrib.auth.views import LogoutView

from django.urls import path

urlpatterns = [

path('logout/', LogoutView.as\_view(), name='logout'),

]

The LogoutView class-based view is used to handle user logout. When a user logs out, they are redirected to the default URL specified by the LOGIN\_REDIRECT\_URL setting.

Customizing Authentication Views

You have the flexibility to customize these views by overriding their attributes or providing custom templates that align with your application’s design aesthetic. Additionally, you can leverage the login\_required decorator or the PermissionRequiredMixin to restrict access to specific views or functionalities based on user permissions or group memberships.

from django.contrib.auth.decorators import login\_required

@login\_required

def profile\_view(request):

# This view can only be accessed by authenticated users

return render(request, 'profile.html')

Password Management

Django includes features for managing user passwords securely, such as password hashing, password validators, and password reset functionality.

1. **Password Hashing**: Django automatically hashes user passwords using the PBKDF2 algorithm before storing them in the database. This ensures that passwords are not stored in plain text, improving security.
2. **Password Reset**: Django provides built-in views and utilities for handling password reset functionality. Users can request a password reset, and Django will send them an email with a link to reset their password.
3. **Password Validators**: Django includes several built-in password validators that enforce password policies. You can use these validators or create custom ones to meet your application’s password requirements.

# settings.py

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

'OPTIONS': {

'min\_length': 9,

}

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

Authentication Views and URLs

Django provides several built-in views and URLs related to user authentication, including login, logout, password reset, and password change views.

Login and Logout Views

pythonCopy codefrom django.contrib.auth.views import LoginView, LogoutView

from django.urls import path

urlpatterns = [

path('login/', LoginView.as\_view(template\_name='registration/login.html'), name='login'),

path('logout/', LogoutView.as\_view(), name='logout'),

]

Password Reset Views

pythonCopy codefrom django.contrib.auth import views as auth\_views

from django.urls import path

urlpatterns = [

path('password\_reset/', auth\_views.PasswordResetView.as\_view(), name='password\_reset'),

path('password\_reset/done/', auth\_views.PasswordResetDoneView.as\_view(), name='password\_reset\_done'),

path('reset/<uidb64>/<token>/', auth\_views.PasswordResetConfirmView.as\_view(), name='password\_reset\_confirm'),

path('reset/done/', auth\_views.PasswordResetCompleteView.as\_view(), name='password\_reset\_complete'),

]

Password Change View

pythonCopy codefrom django.contrib.auth import views as auth\_views

from django.urls import path

urlpatterns = [

path('password\_change/', auth\_views.PasswordChangeView.as\_view(), name='password\_change'),

path('password\_change/done/', auth\_views.PasswordChangeDoneView.as\_view(), name='password\_change\_done'),

]

You can customize these views by overriding their attributes or providing custom templates.

Full Example

Step 1: Enable Django Auth App

Check that the django.contrib.auth and django.contrib.contenttypes apps are in the list of installed apps, if not add them. You can do this by opening [settings.py](http://settings.py) and updating INSTALLED\_APPS

INSTALLED\_APPS = [

...

'django.contrib.auth',

'django.contrib.contenttypes',

...

]

Next make sure the following **middlewares** are present.

MIDDLEWARE = [

...,

'django.middleware.security.SecurityMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

...

]

Step 2: Setting Up Urls & Redirects

Open the [urls.py](http://urls.py) file and add the required accounts urls as shown bellow

from django.urls import path, include

urlpatterns = [

...,

path('accounts/', include('django.contrib.auth.urls')),

path('accounts/profile/',

TemplateView.as\_view(template\_name='accounts/profile.html'),

name='profile'),

path("signup/", SignUpView.as\_view(), name="templates/registration/signup"),

...

]

Here we setup the accounts path which contains the login, logout, password change … etc routes except the signup and profile routes. These are separately added

Next, update the redirect constant variables to redirect to the profile page. Open the [settings.py](http://settings.py) file and update it as bellow

LOGIN\_REDIRECT\_URL = "/accounts/profile"

LOGOUT\_REDIRECT\_URL = "/accounts/profile"

Step 3: Adding Template Files

First create a templates folder at the root of the project and update the TEMPLATES constant in the [settings.py](http://settings.py) file as follows.

TEMPLATES = [

{

...

'DIRS': [ BASE\_DIR / "templates" ],

...

},

]

The following will be the expected folder structure. Assuming you have an app named myapp this can be any app including a dedicated accounts app if you like to separate account related code in to a separate app.

├── db.sqlite3

├── manage.py

├── myapp

│ ├── admin.py

│ ├── apps.py

│ ├── models.py

│ ├── urls.py

│ └── views.py

├── mysite

│ ├── \_\_init\_\_.py

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

└── templates

├── profile.html

├── accounts

│ └── profile.html

└── registration

└── login.html

└── signup.html

profile.html

{% if user.is\_authenticated %} You are logged in as {{ user }}.

<form action="{% url 'logout' %}" method="post">

{% csrf\_token %}

<button type="submit">Log Out</button>

</form>

{% else %} You are not logged in.

<a href="{% url 'login' %}">Click here to log in.</a>

{% endif %}

login.html

{% if form.errors %}

<p>Your username and password didn't match. Please try again.</p>

{% endif %}

<form method="post" action="{% url 'login' %}">

{% csrf\_token %} {{ form.as\_p }}

<input type="submit" value="login" />

</form>

signup.html

{% block title %}Sign Up{% endblock %} {% block content %}

<h2>Sign up</h2>

<form method="post">

{% csrf\_token %} {{ form }}

<button type="submit">Sign Up</button>

</form>

{% endblock %}

Step 4: Adding Signup View

The next thing to do is adding a signup view. These view can be add to your existing app or to a dedicated app for handling accounts. In our case we will add this to our myapp app [views.py](http://views.py) file

from django.contrib.auth.forms import UserCreationForm

from django.urls import reverse\_lazy

from django.views.generic import CreateView

class SignUpView(CreateView):

form\_class = UserCreationForm

success\_url = reverse\_lazy("login")

template\_name = "registration/signup.html"

Step 5: Migrating Your Changes and Running Your Project

The last thing to do is migrate your changes and run your project

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

Practice Exercises

* Implement user registration functionality in your Django application
* Add user login and logout views and URLs
* Customize the built-in authentication views to match your application’s design
* Implement password reset functionality for users
* Define custom permissions and groups to control access to resources in your application

Additional Resources

* [Django Authentication System](https://savanna.alxafrica.com/rltoken/XAWNr7D5_czw0c1GzwYAhg)
* [User Authentication in Django](https://savanna.alxafrica.com/rltoken/yqWUDt-_e43cImYEigE29Q)
* [Authentication Views](https://savanna.alxafrica.com/rltoken/by87HvEez21QiNIA-G-xYA)
* [Password Management](https://savanna.alxafrica.com/rltoken/IdS2aM9iPBackWhNdBlaKA)
* [Permissions and Authorization](https://savanna.alxafrica.com/rltoken/v9dbK629JDvnT1Qjejq53A)
* [Django Login, Logout, Signup, Password Change, and Password Reset](https://savanna.alxafrica.com/rltoken/1tajHMD96BFpiQhtynL5IQ)