Django Authentication System

Cruz Arredondo Jose David, UTT. November 7, 2023

Summary - Django's authentication system is a robust and versatile framework that simplifies the implementation of secure user authentication in web applications. Key components include the user model, authentication views, URL patterns, decorators, and mixins. Developers can customize user models to accommodate specific project requirements, extending the default user model as needed. Authentication views and templates, such as login and password reset, are readily available, enhancing user experience.

I. INTRODUCTION

The authentication system is the gateway to your web application, safeguarding user accounts and sensitive data. A robust authentication system ensures that only authorized users can access specific resources, protecting both user privacy and the integrity of your application. Django's authentication system provides a solid foundation, allowing developers to focus on building innovative features while Django handles the intricate details of user authentication.

II. USER OBJECT

User objects are the core of the authentication system. They typically represent the people interacting with your site and are used to enable things like restricting access, registering user profiles, associating content with creators etc. Only one class of user exists in Django's authentication framework, i.e., 'superusers' or admin 'staff' users are just user objects with special attributes set, not different classes of user objects.

The primary attributes of the default user are:

- username
- password
- email
- first_name
- last_name

```
>>> from django.contrib.auth.models import User
>>> user = User.objects.create_user("john", "lennon@thebeatles.com", "johnpassword")

# At this point, user is a User object that has already been saved
# to the database. You can continue to change its attributes
# if you want to change other fields.
>>> user.last_name = "Lennon"
>>> user.save()
```

The most direct way to create users is to use the included create_user() helper function. [1]

III. CHARACTERISTICS

1) *Authentication Decorators and Mixins:*

Django provides decorators like @login_required and mixins like LoginRequiredMixin to protect views that require authentication.
These decorators and mixins ensure that only authenticated users can access specific views, enhancing security.

2) *Custom Authentication Backends:*

Django allows developers to create custom authentication backends for advanced authentication methods.
Custom backends can be implemented by subclassing django.contrib.auth.backends.ModelBackend and defining custom authentication logic.

```
from django.contrib.auth import authenticate

user = authenticate(username="john", password="secret")
if user is not None:
    # A backend authenticated the credentials
    ...
else:
    # No backend authenticated the credentials
    ...
```

3) *Security Best Practices:*

Django emphasizes security best practices such as secure password hashing, protection against common attacks like Cross-Site Request Forgery (CSRF), and secure session management.
Developers are encouraged to follow these practices to ensure the integrity and confidentiality of user data.

4) *User Models:*

Django provides a built-in user model (User) which can be customized or extended to include additional fields.
Developers can create a custom user model by subclassing AbstractUser, allowing the addition of custom fields and methods.

5) Authentication in web requests

Django uses sessions and middleware to hook the authentication system into request objects.

These provide a request.user attribute on every request which represents the current user. If the current user has not logged in, this attribute will be set to an instance of AnonymousUser, otherwise it will be an instance of User.

```
if request.user.is_authenticated:
    # Do something for authenticated users.
    ...
else:
    # Do something for anonymous users.
    ...
```

IV. LOG AND LOGOUT A USER

If you have an authenticated user you want to attach to the current session - this is done with a login() function.

login(request, user, backend=None)¶
To log a user in, from a view, use login(). It takes an HttpRequest object and a User object. login() saves the user's ID in the session, using Django's session framework.

Note that any data set during the anonymous session is retained in the session after a user logs in.

This example shows how you might use both authenticate() and login():

```
from django.contrib.auth import authenticate, login


def my_view(request):
    username = request.POST["username"]
    password = request.POST["password"]
    user = authenticate(request, username=username, password=password)
    if user is not None:
        login(request, user)
        # Redirect to a success page.
        ...
    else:
        # Return an 'invalid login' error message.
        ...
```

To log out a user who has been logged in via django.contrib.auth.login(), use django.contrib.auth.logout() within your view. It takes an HttpRequest object and has no return value.

```
from django.contrib.auth import logout


def logout_view(request):
    logout(request)
    # Redirect to a success page.
```

V. THE LOGINREQUIREDMIXIN MIXIN

When using class-based views, you can achieve the same behavior as with login_required by using the LoginRequiredMixin. This mixin should be at the leftmost position in the inheritance list.

class LoginRequiredMixin¶
If a view is using this mixin, all requests by non-authenticated users will be redirected to the login page or shown an HTTP 403 Forbidden error, depending on the raise_exception parameter. [2]

You can set any of the parameters of AccessMixin to customize the handling of unauthorized users:

```
from django.contrib.auth.mixins import LoginRequiredMixin


class MyView(LoginRequiredMixin, View):
    login_url = "/login/"
    redirect_field_name = "redirect_to"
```

VI. CONCLUSION

Django's Authentication System stands as a beacon of reliability and flexibility in the realm of web security. Armed with the knowledge gained from this document, developers are well-equipped to fortify their applications, establishing trust with users and fostering a secure online environment. As you venture forward, remember that the key to a successful authentication system lies not just in its complexity, but in its seamless integration, ensuring that users can access their accounts with ease and peace of mind.

VII. REFERENCES

[1] Django (2022) Django, Django Project. Available at: https://docs.djangoproject.com/en/4.2/topics/auth/default/ (Accessed: 07 November 2023).

[2] MozDevNet, M. (2022) Django tutorial part 8: User authentication and permissions - learn web development: MDN, MDN Web Docs. Available at: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Authentication (Accessed: 07 November 2023).