LoginRequiredMixin, Permission and Pagination in Django

Chapter 8

1

LoginRequiredMixin

- The previous chapter discussed on authentication.
- Authentication is the process of registering and logging-in users.
- We might want to limit access to various pages only to logged-in users.
- If a view is using this mixin, all requests by non-authenticated users will be redirected to the login page (settings.LOGIN_URL), passing the current absolute path in the query string.
  Example: /accounts/login/**?next=/articles/new/**

2

LoginRequiredMixin

- Restricting view access is just a matter of adding LoginRequiredMixin at the beginning of the views.
- For example, to limit that only logged-in users can add new post, we import LoginRequiredMixin in the views.py and add it in front of CreateView.

```
# views.py
from django.contrib.auth.mixins import LoginRequiredMixin
...
class ArticleCreateView(LoginRequiredMixin, CreateView):
...
```

- Make sure that the mixin is to the left of CreateView so it will be read first. We want the CreateView to already know we intend to restrict access.

3

login_url

- A logged-out user, on the attempt to access a URL that is mapped to a view with LoginRequiredMixin, will be automatically redirected to the default location for the login page which is at /accounts/login.
- In case your login page is not mapped to this default /accounts/login, use "login_url" to indicate your login page.

```
# views.py
...
class ArticleCreateView(LoginRequiredMixin, CreateView):
        model = models.Article
        template_name = 'article_new.html'
        fields = ['title', 'body',]
        login_url = 'login' #
```

- Alternatively, we can add **LOGIN_URL = 'login'** towards the bottom of settings.py to tell about this. **LOGIN_URL** is the named URL pattern.

4

Redirecting to a passed-in URL

- Whenever a view in Django uses LoginRequiredMixin, you may notice that the URL for the login page contains a URL parameter indicating where the user should be redirected after they log in.
- For example, you might realize something of the form http://youraccount.pythonanywhere.com/accounts/**login?next=/articles/new/** at the URL .
- After logging in, you will be redirected to the URL indicated by the value of the next parameter.
- This is for user-friendliness so that a page that forces users to login will be displayed again right after the successful login, rather than the URL specified by LOGIN_REDIRECT_URL in settings.py.
- So the key thing here is the hidden field with the value of {{ next }}.

5

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

```
<!-- templates/registration/login.html -->
{% extends 'base.html' %}
{% block title %}Login{% endblock %}
{% block content %}
<h2>Login</h2>
<form method="post"> {% csrf_token %}
  {{ form.as_p}}
 <input type="hidden" name="next" value="{{next}}"/>
 <input type="submit" value="Log in">
</form>
<p> If you don't have an account <a href="{% url 'signup' %}">Sign Up here.</a>
{% endblock %}
```

6

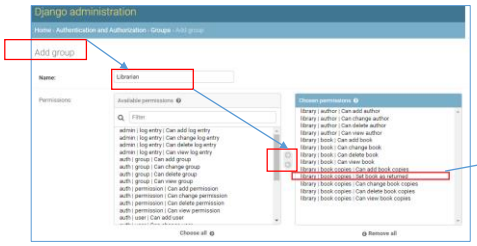
Authentication vs Authorization (permission)

- Authentication is checking user credentials like email and the password is correct.
- Authorization (permission) is testing what an authenticated user can do in the application.
- Permissions are a **rule** (or restrictions) to view, add, change, delete (Django defaults), or custom rules to objects for a specific user or to a group of users.

7

Django's Permissions & Groups

- In Django Admin, you can create a group and assign permissions to it.



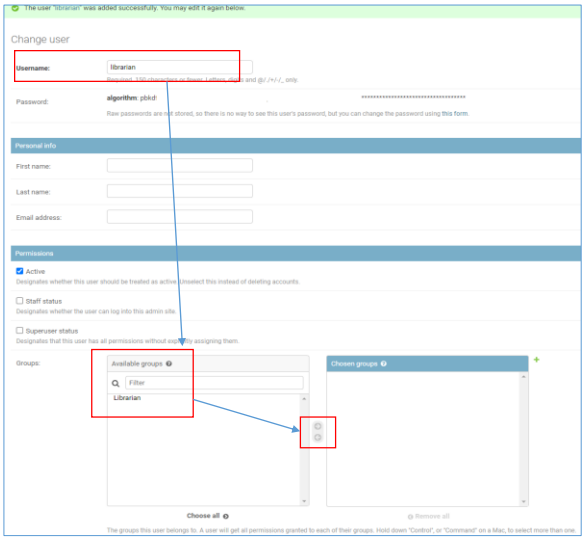
Adding custom permissions:
You can add custom permissions on the model in the "class Meta" section, using the permissions field. Each added permission is defined in a nested tuple containing the permission name and permission display value. E.g.

```
class BookCopies(models.Model):
    ...
    class Meta:
        permissions = (("can_mark_returned", "Set book as returned"),)
```

- By default, every model you defined in models.py comes with 4 kinds of permissions:
  - **add**: Users with this permission can add an instance of the model
  - **delete**: Users with this permission can delete an instance of the model
  - **change:** Users with this permission can update an instance of the model
  - **view:** Users with this permission can view instances of this model
- Permission names follow a very specific naming convention: <app>.<action>_<modelname>

8

Assigning user to the group



9

Testing permissions in templates

- The current user's permissions are stored in a template variable called {{ perms }}.
- You can check whether the current user has a particular permission using the specific variable name within the associated Django "app" — e.g. {{ perms.library }} will be True if the user has all the permissions for the library app, and False otherwise.
- {% if perms.library.add_book %} is to test if the user has the add permission to the book model of the library app.

10

Testing permissions in views

- Permissions can be tested in a class-based view using the PermissionRequiredMixin

```
from django.contrib.auth.mixins import PermissionRequiredMixin
class MyView(PermissionRequiredMixin, CreateView):
        permission_required = 'library.add_book'
```

- For the above example, all requests by users without the required permission return a 403 (HTTP Status Forbidden) exception.

11

Pagination

- For example, on listing the posts, we want to add pagination so that we only list 2 posts on each page. This can be done with setting "paginate_by" attribute in the view.
- This limits the number of objects per page and adds a paginator and page_obj to the context.

```
class ArticleListView(LoginRequiredMixin, ListView):
    model = models.Article
    template_name = 'article_list.html'
    paginate_by = 2
    login_url = 'login'
```

12

paginator and page_obj

Having set “paginate_by” attribute in the view, we can then make use of the paginator and page_obj in our template files, such as:

- page_obj.has_previous, page_obj.has_next: Boolean
- page_obj.previous_page_number, page_obj.next_page_number : an integer
- page_obj.number: an integer, the current page numner
- page_obj.paginator.page_range
- page_obj.paginator.num_pages: an integer, the total number of pages

```
<div>
    {% if page_obj.has_previous %}
       <a href="?page=1">&laquo; first</a>
       <a href="?page={{ page_obj.previous_page_number }}">previous</a>
    {% endif %}

    {% for page in page_obj.paginator.page_range %}
       {% if page == page_obj.number %}
          {{page}}
       {% else %}
          <a href="?page={{page}}">{{page}}</a>
       {% endif %}
    {% endfor %}

    {% if page_obj.has_next %}
       <a href="?page={{ page_obj.next_page_number }}">next</a>
       <a href="?page={{ page_obj.paginator.num_pages }}">last &raquo;</a>
    {% endif %}
</div>
```

13