SDEV 2401

Rapid Backend App Development

Authentication Authorization and Users - 3 and 4

Expectations - What I expect from you

- No Late Assignments

- No Cheating

- Be a good classmate

- Don't waste your time

- Show up to class

Agenda

On the right is what we will cover today.

- What is Authorization?
- Authentication vs Authorization — Theory
- Role-Based Access Control (RBAC) — New Example
- Authorization in Django — Approaches
- Custom Permission Checks — Example
- Associating Data with Users
- Permissions & Groups — Theory and Example
- Template Authorization — Show/Hide UI
- Custom Error Pages
- Challenge — Events Example
- Summary

What is Authorization?

- Authorization is the process of determining what an authenticated user is allowed to do within the system.
- Many times people will call this Role-Based Access Control (RBAC).
- This is normally based on user roles, permissions, or specific attributes.

For example:

- An admininstrator can manage users and settings.
- A moderator can manage content.
- A regular user can only view and edit their own profile.
- A guest user may have limited access to certain features.

This is a cruicial part of application security and user experience.

Authentication vs Authorization — Theory

Authentication: Proves user identity (e.g., login with email & password). **Authorization**: Determines what an authenticated user can do (e.g., access admin panel).

- **Authentication** answers: "Who are you?"
- **Authorization** answers: "What are you allowed to do?"

Example:

- Authentication: A library member logs in with their card number.
- Authorization: Only librarians can add new books; members can only borrow.

Role-Based Access Control (RBAC) Example

RBAC assigns permissions based on user roles.

- **Roles**: e.g., "Editor", "Author", "Reader"
- **Permissions**: e.g., "edit_article", "publish_article", "view_article"

Example:

- An "Author" can write and edit their own articles.
- An "Editor" can approve and publish any article.
- A "Reader" can only view published articles.

Authorization in Django — Approaches

There’s two main levels of authorization in Django:

- **Model-Level**: Controls access to data (e.g., only "Editors" can delete articles).
- **View-Level**: Controls access to endpoints (e.g., only "Authors" can access `/submit-article/`).

Django Tools:

- Decorators: `@login_required`, `@user_passes_test`, `@permission_required`
- Mixins: `UserPassesTestMixin`, `PermissionRequiredMixin`, we haven’t talked about this yet but we’ll cover this in a later module when we do class based views.

Custom Permission Checks — Example

You can use a *custom function* for view-level authorization, and use the `@user_passes_test` decorator which takes a function that returns `True` or `False`.

- if `True`, the user is allowed access.
- if `False`, they are denied access and redirected to the login page (more on this later).

```python
def is_editor(user):
    return user.role == 'Editor'

@login_required
@user_passes_test(is_editor, login_url='login')
def publish_article(request):
    # Only editors can access
    ...
```

What does this do? - if the user is not logged in, they are redirected to the login page. - if logged in but not an editor, they will be redirected to the login page as well (more on this later). - if logged in and an editor, they can access the `publish_article` view.

Custom checks allow fine-grained control, e.g., only users with a certain attribute or group.

Associating Data with Users

User-specific data: Tie records to the user who created them.

Example:

- Each "Author" sees only their own drafts.
- "Editors" see all drafts.

Django Pattern to add a user:

- Add a `created_by` field (ForeignKey to `User` or `settings.AUTH_USER_MODEL`) in your model.
- In the Form don’t include the `created_by` field.
- In the View when processing the form:
  - use `form.save(commit=False)` to get the instance without saving.
  - Set `form.created_by = request.user` when saving.

We’ll see more of this in the example.

Permissions & Groups (Role Based Access Control)

Django has a built in permissions and groups system that can be used to implement Role based Access Control on your models.

Permissions: Define what actions a user can perform (add, change, delete, view). Every model gets these by default, and will have the following name conventions

- `APPNAME.add_MODELNAME` to create
- `APPNAME.change_MODELNAME` to update
- `APPNAME.delete_MODELNAME` to delete
- `APPNAME.view_MODELNAME` to view More on this in the example

Groups: Bundle permissions and assign to users based on their role. You'll be doing this normally in the admin interface.

Permissions & Groups (Continued)

Example:

- "Moderators" group: can delete comments.
- "Members" group: can add comments.

Django: Assign users to groups in the admin, or programmatically:

```
from django.contrib.auth.models import Group
user.groups.add(Group.objects.get(name='Moderators'))
```

We'll take a look at this in the example as well.

Template Authorization — Show/Hide UI

Templates can check permissions to show/hide elements.

Example:

```
{% if perms.blog.add_post %}
    <a href="{% url 'add_post' %}">Add Post</a>
{% endif %}
```

- Only users with the `blog.add_post` permission see the link. This would be for a `Post` model in an app called `blog`.

Important: You can also use a custom model property like `membership_level` that you added to the user model in the last example.

```
{% if user.membership_level == 'premium' %}
    <a href="{% url 'premium-content' %}">Premium Content</a>
{% endif %}
```

Custom Error Pages

Custom error pages improve UX for forbidden actions. The only thing that you need to do is implement the error page template. Available error pages to customize are:

- 400 Bad Request: `400.html`
- 403 Forbidden: `403.html`
- 404 Not Found: `404.html`
- 500 Internal Server Error: `500.html`

Django:

- Create `403.html` in your templates.
- Use `@permission_required(..., raise_exception=True)` to trigger.

Example: A user tries to access `/publish-article/` without permission and sees a friendly "403 Forbidden" page.

Challenge — Events Example

Model:

- `Event` : name, date, created_by

Permissions:

- Only "Teachers" can create/update/delete events.
- "Students" can only view.

Views:

- List Events: all logged-in users
- Create Event: only "Teachers"

Summary

- Authentication: Who are you?
- Authorization: What can you do?
- RBAC: Assign permissions by role.
- Django: Use decorators, mixins, permissions, and groups.
- Always associate user-specific data for accountability.
- Use templates and custom error pages for better UX.

Example

Let’s go do an example together