




Welcome to the CoGrammar

Tutorial: Advanced Django

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.



Software Engineering Session Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(Fundamental British Values: Mutual Respect and Tolerance)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

Software Engineering Session Housekeeping cont.

- For all **non-academic questions**, please submit a query:
www.hyperiondev.com/support
- Report a **safeguarding** incident:
www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

- **Completion:** All mandatory tasks, including Build Your Brand and resubmissions by study period end
- **Interview Invitation:** Within 4 weeks post-course
- **Guided Learning Hours:** Minimum of 112 hours by support end date (10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

- **Final Job or Apprenticeship Outcome:** Document within 12 weeks post-graduation
- **Relevance:** Progression to employment or related opportunity

Learning Outcomes

- Implement a basic user registration form using Django forms.
- Implement Django's authentication views to perform user login and registration
- Explain what Permissions are in Django
- Create Permissions in their Django projects.
- Assign Permissions to a user in their projects.

Learning Outcomes

- Explain what Groups are in Django.
- Create Groups in their Django projects.
- Assign Permissions to Groups.
- Assign Group to a user to apply Permissions of Group to the user.

**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Authentication

April 2024

Overview of Django's Authentication System

- Django provides a robust authentication system with several built-in components to manage user authentication.
- Components include the:
 - User Model
 - Authentication views
 - Authentication backends

Overview: User Model

- The User model in Django is **part of the `django.contrib.auth` module** and represents user accounts in a Django application.
- It **includes fields for storing essential user information** such as username, password, email, and other personal details.
- The User model also **supports methods for creating, updating, and authenticating** users.

Overview: Authentication Views

- Django provides a set of built-in views for handling user authentication tasks, such as login, logout, and password management.
- These views are part of the `django.contrib.auth.views` module and are designed to be used out-of-the-box, reducing the need for developers to write custom authentication logic.

Auth Views: Key Views

- **LoginView**: Handles user login. This view renders a login form and processes user credentials to authenticate users.
- **LogoutView**: Logs users out by terminating their session.
- **PasswordChangeView**: Allows users to change their password while logged in.
- **PasswordResetView**: Provides functionality for users to reset their password if forgotten, typically involving sending a reset link to their email.

Auth Views: Code Example

- In urls.py:

```
from django.urls import path
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
    path('password_change/', auth_views.PasswordChangeView.as_view(), name='password_change'),
    path('password_reset/', auth_views.PasswordResetView.as_view(), name='password_reset'),
]
```

Creating Login and Logout Views

- For more control, you can create custom views.

```
from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect

def custom_login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)
        if user is not None:
            login(request, user)
            return redirect('home')
    return render(request, 'login.html')

def custom_logout(request):
    logout(request)
    return redirect('home')
```

URL Configuration

- Map your views to URLs to make them accessible.

```
from django.urls import path
from .views import register, custom_login, custom_logout

urlpatterns = [
    path('register/', register, name='register'),
    path('login/', custom_login, name='login'),
    path('logout/', custom_logout, name='logout'),
]
```


**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Permissions

April 2024

Permissions

- Our applications have **multiple tables** in our databases and we would like to **limit** who has **access** where.
- Permission will allow us to do just that.
- We can set a **permission** for a **specific table** and only allow a **user** with the **correct** permissions to **access** or change the table.
- When having a table **Users** we can create a permission "**can edit users**" then only **users with** this **permission** can edit users in our table.

Django Permissions

- Luckily for us we don't have to build this permission system from scratch, although you can.
- Django has a **built-in** permissions system we can make use of.
- We can use **permission** with our **models** to **limit access** to the data within that model's database table.

Django Permissions

- Having “`django.contrib.auth`” within our **installed apps** we get four default permissions for each of our models.
 - Add
 - Change
 - Delete
 - View
- These **permissions** are **created** when calling:
 - `python manage.py migrate`

Django Permissions

- Assuming you have an application with an app_label **blog_app** and a **model** named **Blog**, to test for basic permissions you should use:
 - **add:** `user.has_perm('blog_app.add_blog')`
 - **change:** `user.has_perm('blog_app.change_blog')`
 - **delete:** `user.has_perm('blog_app.delete_blog')`
 - **view:** `user.has_perm('blog_app.view_blog')`

Adding Permissions

To add specific permissions to a user we first have to get the permissions.

```
content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.get(
    codename="can_publish",
    content_type=content_type,
)
```


Adding Permissions

Then we can take the permission and add it to our user.

```
user.user_permissions.add(permission)
```

Finally we can check if the user has the correct permissions.

```
user.has_perm("myapp.change_blogpost")
```

Permissions

We can restrict user from accessing certain views based on their permissions using the `permission_required()` decorator.

```
from django.contrib.auth.decorators import permission_required

@permission_required("polls.add_choice")
def my_view(request):
```

CoGrammar

Groups

April 2024

Groups

- Now that we have permissions we would like to give **multiple users** the **same** set of permissions.
- It would be very tedious to have to go to each individual user and add each permission.
- This is where groups come into play.
- We can create a **group** and **give** it a **set** of **permissions**.
- When we then want to **assign** those **permissions** to a **user** we can just **assign** them to the **group**

Groups

- We saw that we had the option to **create** permission using **code** or we could make use of the Django **admin site**.
- The **same** applies for **groups** we can **create** and **assign** permissions to a group using **code** or we could do it through the **admin site**.
- The Group Model class has 2 attributes:
 - Name
 - Permissions

Creating Groups

```
from django.contrib.auth.models import Group, Permission
from django.contrib.contenttypes.models import ContentType

from .models import BlogPost

new_group, created = Group.objects.get_or_create(name='publishers')

content_type = ContentType.objects.get_for_model(BlogPost)
permission = Permission.objects.get(
    codename="change_blogpost",
    content_type=content_type,
)
new_group.permissions.add(permission)
```


Summary

- **Authentication:** User authentication is fundamental for securing access to web applications, ensuring that users are who they claim to be. Django provides a robust, built-in authentication system that includes models and views to handle user authentication seamlessly.
- **Data security:** We have a responsibility to protect the data of our users. We have to limit access we provide users within our applications.
- **Permissions:** We can create and add permissions to our web application to limit users access to certain parts of the program as well as their access to the data contained within our program.
- **Groups:** We can use groups to apply a broad set of permissions to a user belonging to a specific group.

Questions and Answers



Thank you for attending



Department
for Education

CoGrammar

