Welcome to this CoGrammar session:

Django Authentication

The session will start shortly...

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

CoGrammar

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
8-Week 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

CoGrammar

Learning Outcomes

- Identify the purpose of user authentication in web applications.
- Explain the role of Django's built-in authentication system.
- Implement a basic user registration form using Django forms.
- Differentiate between Django's authentication views and custom views for user login and registration.
- Assess the security implications of handling user authentication data.
- Design a complete authentication system with user registration, login, and logout functionality.

Polls

- *Refer to the polls section to vote for you option.*

1. **How familiar are you with the concept of user authentication in web applications?**
   a. Very familiar
   b. Somewhat familiar
   c. Heard of it but don't know much
   d. Not familiar at all

Polls

- *Refer to the polls section to vote for you option.*

2. **Have you ever implemented authentication features (such as login and registration) in a Django project?**

   a. Yes, multiple times

   b. Yes, once or twice

   c. No, but I've used Django for other purposes

   d. No, I have never used Django before

SKILLS FOR LIFE

SKILLS BOOTCAMPS

Department for Education

CoGrammar

Django Authentication

May 2024

Importance of User Authentication

- User authentication is the process of verifying the identity of users before granting them access to secured resources.
- Authentication ensures that sensitive data is only accessible to authorised users.
- The Verizon 2023 Data Breach Investigations Report revealed that 61% of breaches involved compromised user credentials, often due to social engineering attacks or weak password practices *(Expert Insights)*.

Recent Authentication Breaches

- Bank of America experienced a significant breach due to vulnerabilities in a third-party system. This breach compromised sensitive customer data, including names, social security numbers, and account details, affecting over 57,000 individuals *(Techopedia)*.

- Another notable breach involved Trello, where poor API security allowed unauthorised access to user data. This incident affected over 15 million users, exposing email addresses, names, and usernames, which were later sold on a hacking forum *(Techopedia)*.

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

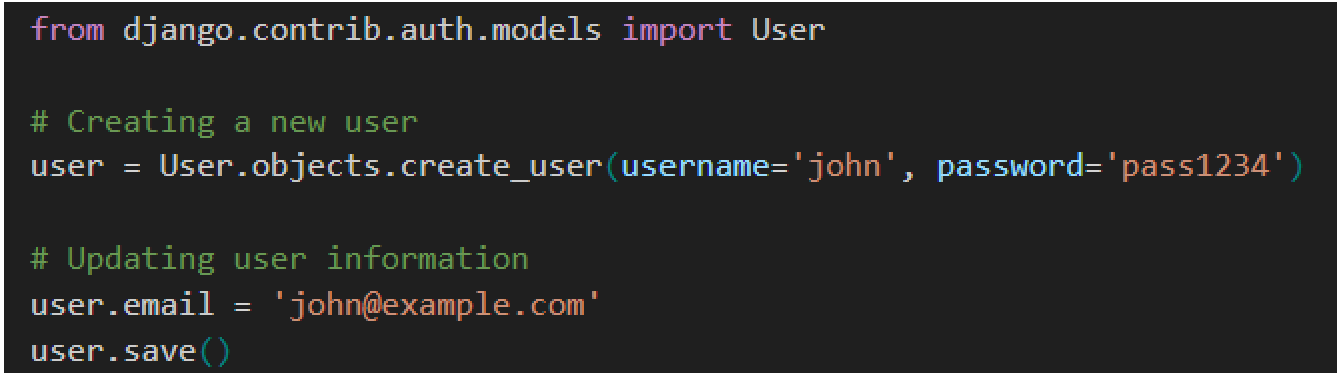
- The User model in Django is part of the django.contrib.auth module and represents user accounts in a Django application.
- It includes fields for storing essential user information such as username, password, email, and other personal details.
- The User model also supports methods for creating, updating,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

User Model: Key Features

- Fields: The default User model includes fields like username, password, email, first_name, and last_name.
- Authentication: The model includes methods for setting and checking passwords and performing authentication checks.
- Custom User Models: Django allows customisation of the User model by either extending the existing model or substituting it entirely with a custom model using the AUTH_USER_MODEL setting.

User Model: Code Example

- No additional code is needed to implement the default User model provided by Django.
- To apply the User Model in Python Code, we add a function in the views.py. The below logic will then be part of a function.



```python
from django.contrib.auth.models import User

# Creating a new user
user = User.objects.create_user(username='john', password='pass1234')

# Updating user information
user.email = 'john@example.com'
user.save()
```

CoGrammar

Overview: Authentication Views

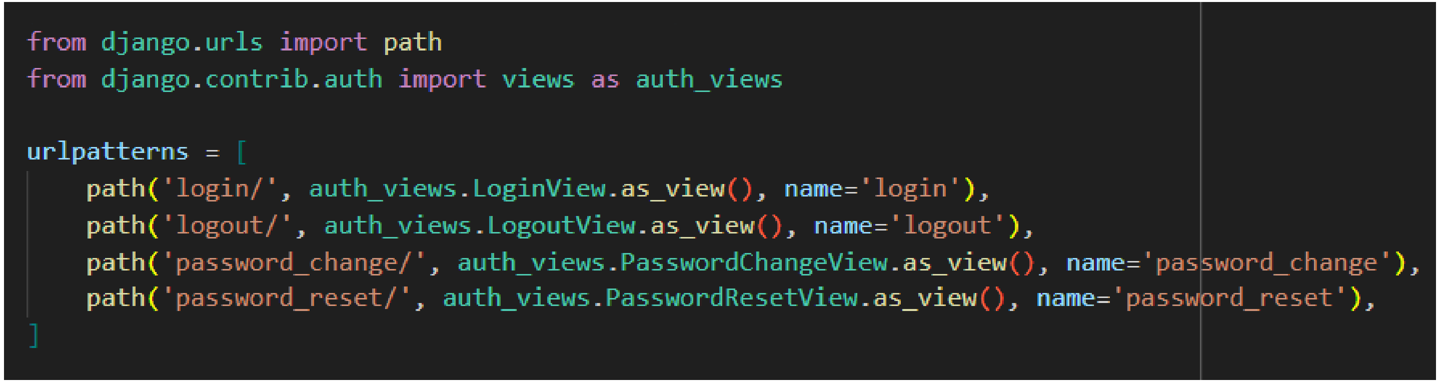
- Django provides a set of built-in views for handling user authentication tasks, such as login, logout, and password management.
- These views are part of the django.contrib.auth.views module and are designed to be used out-of-the-box, reducing the need for developers to write custom authentication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



```python
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'),
]
```

Overview: Authentication Backends

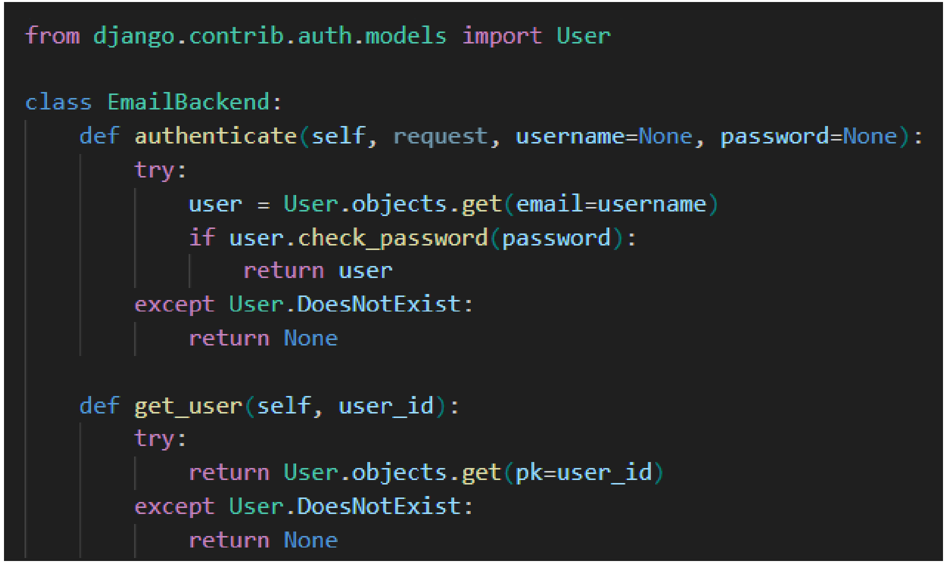
- Authentication backends in Django are responsible for authenticating users by verifying their credentials against a data source.
- A backend is a class that implements two methods: authenticate and get_user.
- Django uses these backends to process login requests and to retrieve user information.

Auth Backends: Key Features

- Default Backends: Django includes a default authentication backend that verifies user credentials against the User model stored in the database.
- Custom Backends: Developers can create custom authentication backends to integrate with external systems or to implement custom authentication logic.

Auth Backends: Code Example

- Creating a custom backend in backends.py:



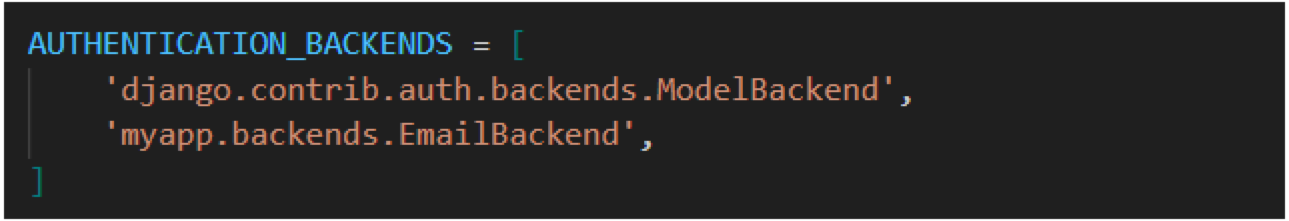
```python
from django.contrib.auth.models import User

class EmailBackend:
    def authenticate(self, request, username=None, password=None):
        try:
            user = User.objects.get(email=username)
            if user.check_password(password):
                return user
        except User.DoesNotExist:
            return None

    def get_user(self, user_id):
        try:
            return User.objects.get(pk=user_id)
        except User.DoesNotExist:
            return None
```

Auth Backends: Code Example

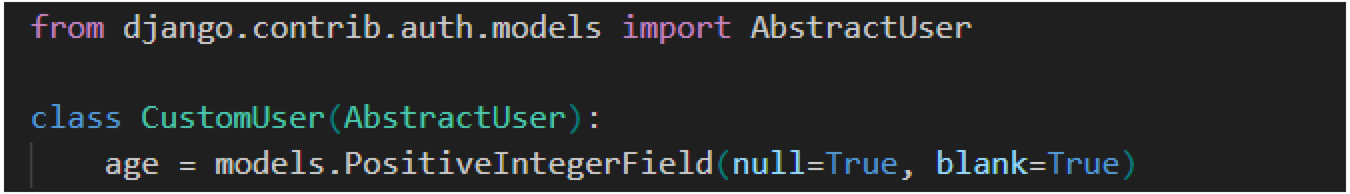
- In settings.py:



```python
AUTHENTICATION_BACKENDS = [
    'django.contrib.auth.backends.ModelBackend',
    'myapp.backends.EmailBackend',
]
```

Creating a Custom User Model

- Django's default User model includes fields for username, password, email, and more.
- You can extend the User model to include additional fields specific to your application. CustomUser inherits from AbstractUser and then additional fields can be added.
- In models.py:

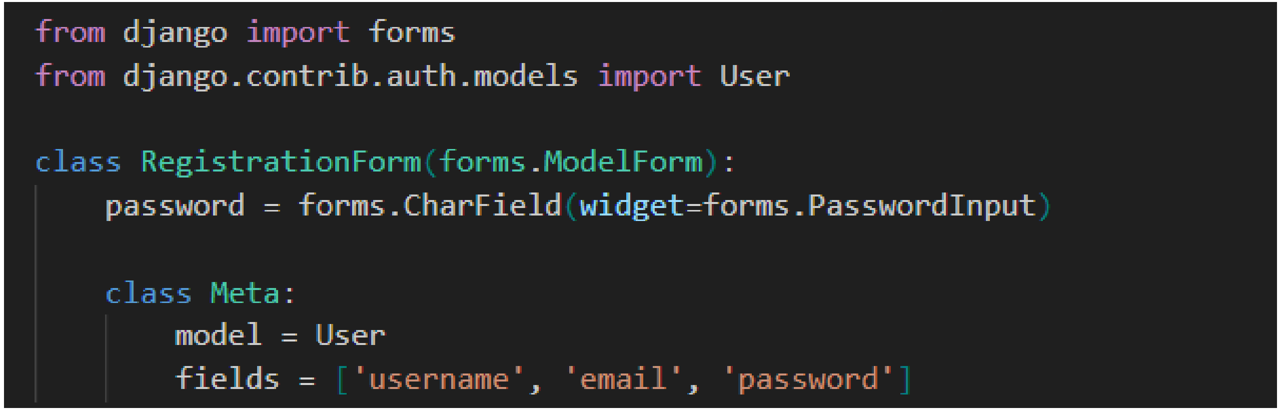


```python
from django.contrib.auth.models import AbstractUser

class CustomUser(AbstractUser):
    age = models.PositiveIntegerField(null=True, blank=True)
```

Building a Registration Form

- Django forms make it easy to create a user registration form.
- Define the form class, handle form submission, and save user data securely.



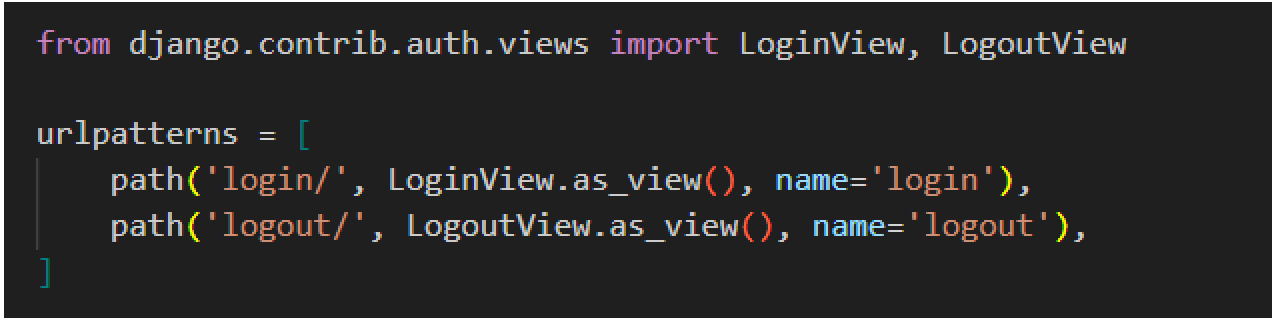
```python
from django import forms
from django.contrib.auth.models import User

class RegistrationForm(forms.ModelForm):
    password = forms.CharField(widget=forms.PasswordInput)

    class Meta:
        model = User
        fields = ['username', 'email', 'password']
```

Creating Login and Logout Views

- Django’s LoginView and LogoutView provide ready-made views for user authentication.

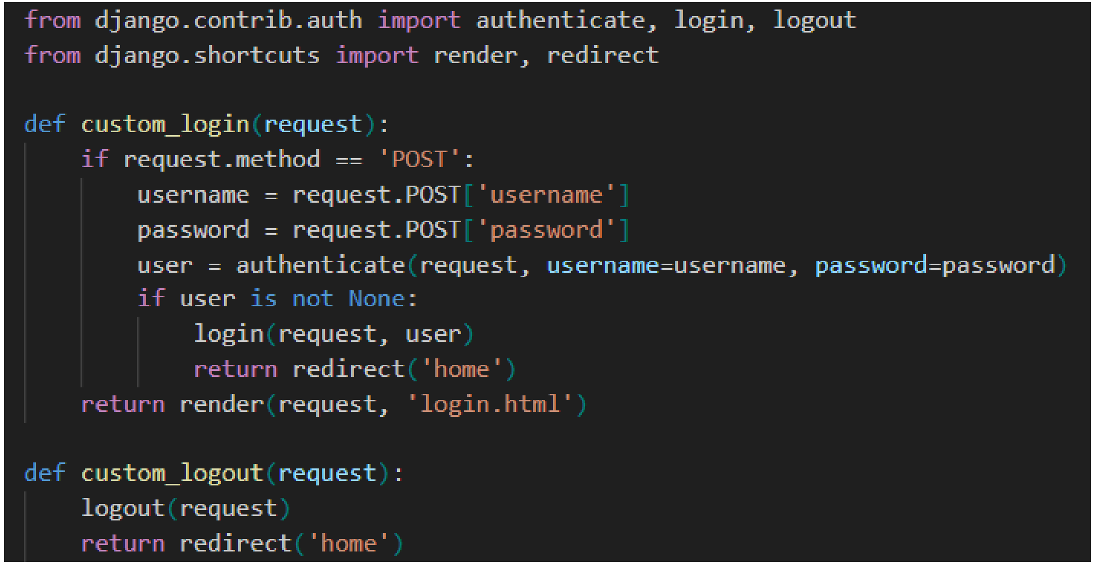


```python
from django.contrib.auth.views import LoginView, LogoutView

urlpatterns = [
    path('login/', LoginView.as_view(), name='login'),
    path('logout/', LogoutView.as_view(), name='logout'),
]
```

Creating Login and Logout Views

- For more control, you can create custom views.



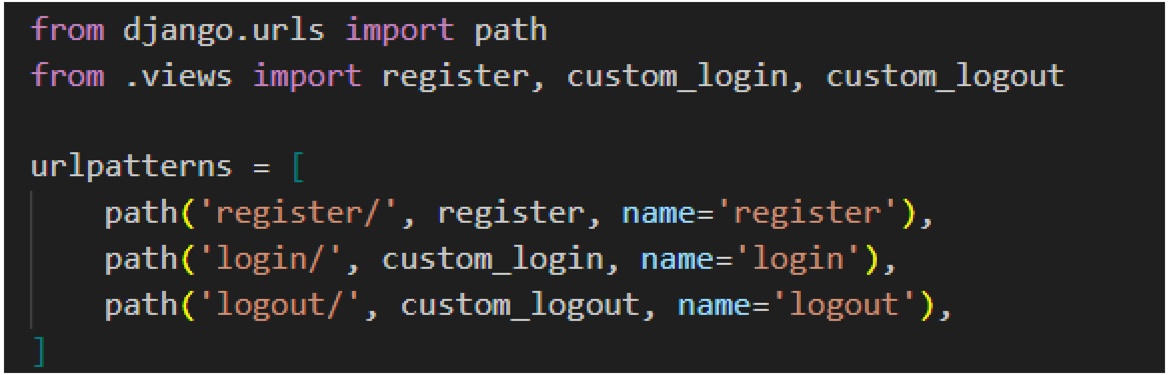
```python
from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect

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.

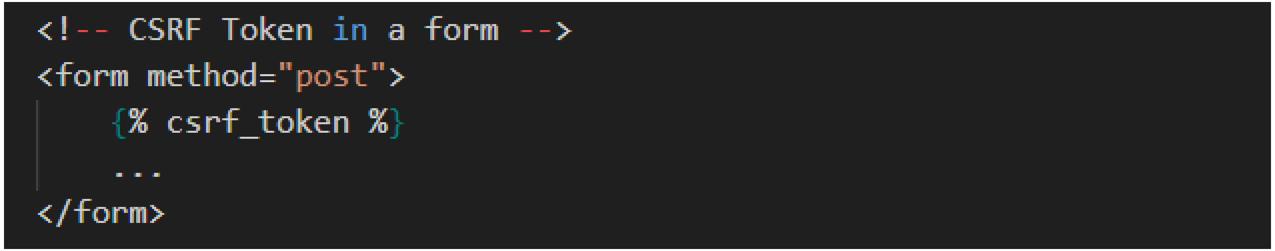


```python
from django.urls import path
from .views import register, custom_login, custom_logout

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

Enhancing Security

- Implement CSRF protection to prevent cross-site request forgery.



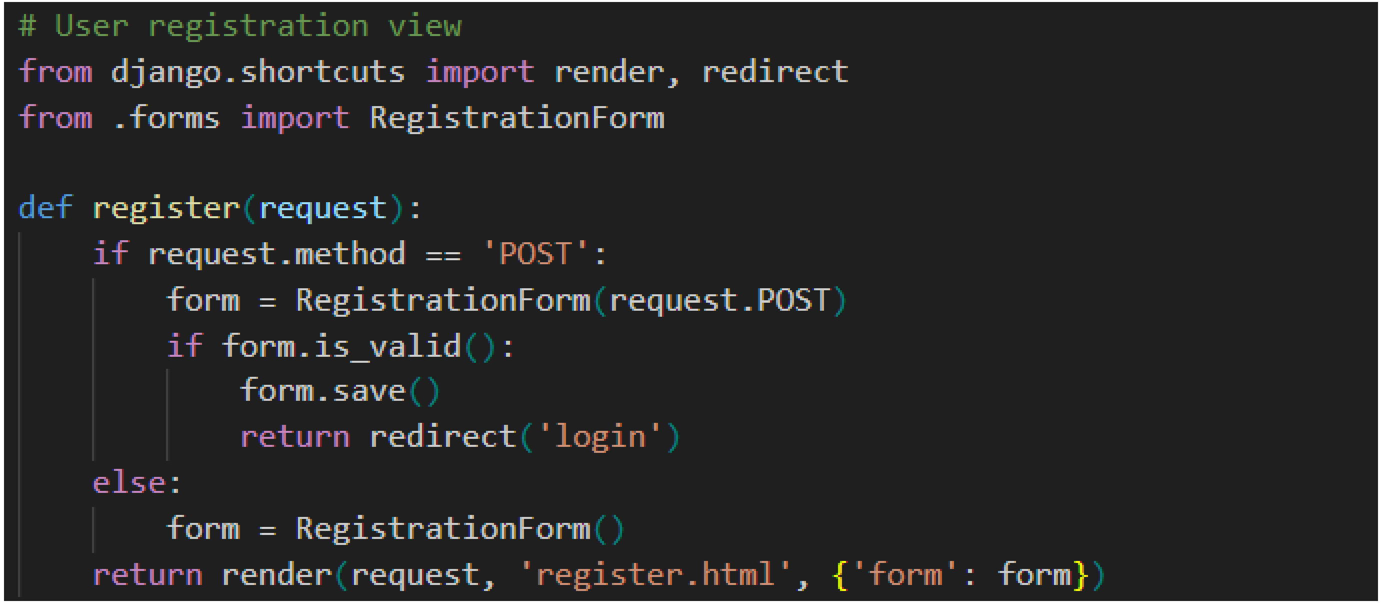
```
<!-- CSRF Token in a form -->
<form method="post">
    {% csrf_token %}
    ...
</form>
```

- Ensure passwords are securely hashed (formatted) before storing. Django automatically hashes passwords when you create or update a user using the built-in User model.
- Manage user sessions securely.

Building a Complete Authentication System

- Combine registration, login, and logout views into a cohesive authentication system.
- Ensure all components are properly integrated and tested.

Building a Complete Authentication System



```python
# User registration view
from django.shortcuts import render, redirect
from .forms import RegistrationForm

def register(request):
    if request.method == 'POST':
        form = RegistrationForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('login')
    else:
        form = RegistrationForm()
    return render(request, 'register.html', {'form': form})
```

Let’s take a short break

CoGrammar

Let’s get coding!

CoGrammar

Polls

- *Refer to the polls section to vote for you option.*

1. **What is the primary purpose of user authentication in web applications?**

    a. To personalise the user experience

    b. To verify the identity of users before granting access to secured resources

    c. To enhance the visual appeal of the application

    d. To improve the performance of the application

Polls

- *Refer to the polls section to vote for you option.*

2. Which of the following Django views is used to handle user login functionality?

    a. RegisterView

    b. LogoutView

    c. LoginView

    d. ProfileView

Questions and Answers

CoGrammar

Summary

- 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.
- Our authentication functionality included:
  - Project creation and setup
  - Creating and configuring registration, login, and logout views
  - Integrating these views with URLs and templates

CoGrammar

Summary

- We also emphasised the need to hash passwords securely, protect against common vulnerabilities, and follow best practices to ensure data integrity and confidentiality.
- In our demonstration, we had a look at:
  - Setting up a new Django project and application
  - Creating Registration, Login, and Logout Views
  - Practical integration of views with URL configurations and HTML templates.

Homework

- User Profile Management:
  - Create views and templates for users to manage their profiles, such as updating their email address, password, or profile information.
  - Implement forms and validation logic to handle profile updates securely.

Thank you for attending

SKILLS FOR LIFE
SKILLS BOOTCAMPS

Department for Education

CoGrammar