NIET Greater Noida
Autonomous Institute

Course code ACSE0512
Course Name : Python web development with Django

Python Web Development with Django

Unit: III

Integrating Accounts & Authentication on Django

Course Details
(B. Tech. 5th Sem)

Shalini Shrotriya
(Asst. Professor)
IT Department

UNIT-III: Integrating Accounts & Authentication on Django

Introduction to Django Authentication System, Security Problem & Solution with Django Creating Registration Form using Django, Adding Email Field In Forms, Configuring email settings, Sending emails with Django, Adding Grid Layout On Registration Page, Adding Page Restrictions, Login Functionality Test and Logout.

Course code ACSE0512
Course Name : Python web development with Django

B. TECH (IT)
Evaluation Scheme

Session 2020-21	Third Year	SEMESTER V	ESC (3)	PCC (14)	ELC (6)	PW (1)

Sl. No.	Subject code	Subject	Periods			Evaluation Schemes				End Semester		Total	Credit	Course Type
			L	T	P	CT	TA	TOTAL	PS	TE	PE			
1		Design Thinking -II	2	1	0	30	20	50		100		150	3	ESC
2	20CS501	Database Management System	3	1	0	30	20	50		100		150	4	PCC
3	20CS502	Web Technology	3	0	0	30	20	50		100		150	3	PCC
4	20CS503	Compiler Design	3	1	0	30	20	50		100		150	4	PCC
5		Department Elective I	3	0	0	30	20	50		100		150	3	ELC
6		Department Elective II	3	0	0	30	20	50		100		150	3	ELC
7	P20CS501	Database Management System Lab	0	0	2				25		25	50	1	PCC
8	P20CS502	Web Technology Lab	0	0	2				25		25	50	1	PCC
9	P20CS503	Compiler Design Lab	0	0	2				25		25	50	1	PCC
10		Internship Assessment	0	0	2				50			50	1	PW
11		Constitution of India / Essence of Indian Traditional Knowledge	2	0	0	30	20	50		50		100	0	NC
12		MOOCs for Honors degree												

Unit II Objective

- Introduction to Django Authentication System.
- Security Problem & Solution with Django
- Creating Registration Form using Django.
- Adding Email Field In Forms, Configuring email settings, Sending emails with Django.
- Adding Grid Layout On Registration Page.
- Adding Page Restrictions, Login Functionality Test and Logout

Unit III Objective

Topic : Introduction to Django Authentication System.

In this topic, the students will gain to giving users the ability to create an account they can sign into is a common function for many websites.

Introduction to Django Authentication System

The Django authentication system handles both **authentication** and **authorization**.

Authentication verifies a user is who they claim to be.

Authorization determines what an **authenticated user** is allowed to do.

Some **authentication options** Django framework provide

- Basic authentication
- Token authentication
- Session authentication
- JSON Web Token authentication
- OAuth 2.0 authentication
- Custom authentication

Introduction to Django Authentication System

This module provides the core of the authentication system, including the user model and authentication-related views.

Authentication support is bundled as a **Django contrib module** in django.contrib.auth.

By default, the required configuration is already included in the **settings.py** by the command

django-admin startproject projectname

It reflect in **settings.py** section **INSTALLED_APPS**

django.contrib.auth - core authentication framework and its default models

django.contrib.contenttypes - allows permissions to be associated with models you create

Introduction to Django Authentication System

Two more in middleware

SessionMiddleware - Manages sessions across requests

AuthenticationMiddleware - Associates users with requests using sessions.

Core components- User objects are used to represent registered users.

Introduction to Django Authentication System

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.

Django comes with a **user authentication system**. It handles user **accounts**, **groups**, **permissions** and **cookie-based user sessions**

in next slide we discuss a attributes of User Model

Introduction to Django Authentication System

The **primary attributes** of the default **User Model**

Required fields : username, password

Optional fields : email, first_name, last_name

Boolean flag fields are

is_staff: To check the user can access the admin interface.

is_active: To check whether this account is currently active.

is_superuser: Gives the user all permissions without **explicitly assigning** them.

last_login: Date and time of the user's last login.

date_joined: Date and time when the user account was created.

We can create custom user models if needed

Introduction to Django Authentication System

Steps to create a new user

1. Import the User Model
2. User **create_user() method**

Two methods to create new user

1. Using Django **shell** script
2. By create **HTML** Interface

Introduction to Django Authentication System

By using Django Shell script

```
# command to activate shell script
python manage.py shell
```

- from django.contrib.auth.models import User
- user = User.objects.create_user(username = 'user@123', password = 'A@123456')
- user.save()
- print(user.username)

Introduction to Django Authentication System

Now we want to add or update data to user fields after creating user below command to add email value in user

- **user.email = 'example@yahoo.com'**
- **user.save()**

Introduction to Django Authentication System

By using html interface, views and urls

1. Make a template for user registration.
2. Create a view to register a new user in **views.py**.
3. Make a router in **urls.py**.
4. Place you static folder detail in **settings.py**.

Security in Django

Cross site scripting (XSS) protection

XSS attacks allow a user to inject client side scripts into the browsers of other users.

This is usually achieved by storing the malicious scripts in the database where it will be retrieved and displayed to other users, or by getting users to click a link which will cause the attacker's JavaScript to be executed by the user's browser

XSS attacks are origin from un trusted source(like cookies and web services)

Security in Django

this code cause **(XSS attack)**: {{ user_input }}

if **user_input** contains **<script>alert('XSS')</script>**

This will render the malicious input cause the XSS attacks

For this attack Django provide the built-in protection from XSS.

Django **automatically** escapes special characters like <, >, and &
when **rendering templates**. This **prevents** user-provided data **from being** treated **as HTML/JavaScript.**

if you use the template in Django : above code render like

** <script>alert('XSS');</script> **

Security in Django

Be careful when using **is_safe, mark_safe, or turning off autoescape.**

Cross-Site Request Forgery (CSRF) Protection:

- CSRF allows **malicious users** to **execute actions** using **another user's credentials**.
- Django's CSRF protection checks for a **secret token** in each **POST request**.
- Using **HTTPS** enhances **CSRF protection** by **validating** the **HTTP** referrer **header**.
- Be careful when exempting views from CSRF protection

Security in Django

In this example **user session data** is used to **perform actions** without their **consent**.

```
<form action="http://example.com/comment" method="POST">
        <input type="hidden" name="comment" value="Nice post!"/>
        <input type="submit" value="Submit"/>
 </form>
```

Django automatically adds a **CSRF token** to forms, which **must** be **verified** on the **server side**:

```
<form method="post"> {% csrf_token %}
    <input type="text" name="comment">
     <input type="submit" value="Submit">
 </form>
```

Security in Django

SQL Injection attack:

Allowing direct user input in raw SQL queries

```
cursor.execute("SELECT * FROM users WHERE username = '%s'" % user_input)
```

in this if user_input or 1=1 it can **allow attackers** to **bypass authentication**

To project use Django's **ORM**, which automatically escapes input

```
User.objects.filter(username=user_input)
```

Above example provide a **parameterized** query to **protection** from **SQl injections.**

Clickjacking

- Clickjacking is a type of attack where a **malicious site wraps** another site in a **frame**.
- This attack can result in an **unsuspecting** user being tricked into performing **unintended actions** on **the target site**

Example :

```
<iframe src="http://example.com" style="opacity: 0; width: 100%; height: 100%;"></iframe>
```

Django provide **protection middleware adds** this **header** to **prevent framing.**

X-Frame-Options: DENY

Above code ensures your site cannot be **embedded** in **frames** by **third-party websites.**

Security in Django

SSL/HTTPS

- **Without HTTPS**, attackers can **eavesdrop** on **traffic**, **capturing sensitive** data such as login credentials. Url not using http

http://example.com/login

Django can **redirect all HTTP** requests to **HTTPS**

SECURE_SSL_REDIRECT = True

Required for above command (must check)

SESSION_COOKIE_SECURE = true

CSRF_COOKIE_SECURE = true

To protect cookies

Security in Django

SSL/HTTPS

- **Without HTTPS**, attackers can **eavesdrop** on **traffic**, **capturing sensitive** data such as login credentials. Url not using http

http://example.com/login

Django can **redirect all HTTP** requests to **HTTPS**

SECURE_SSL_REDIRECT = True

Required for above command (must check)

SESSION_COOKIE_SECURE = true

CSRF_COOKIE_SECURE = true

To protect cookies

Security in Django

Host Header Validation

- An attacker modifies the Host header to point to a **malicious site**.

Host: evilsite.com

Django ensures that only trusted hostnames are allowed

ALLOWED_HOSTS = ['example.com']

Django validates Host headers against the **ALLOWED_HOSTS** setting in the **django.http.HttpRequest.get_host()** method.

Security in Django

Host Header Validation

- An attacker modifies the Host header to point to a **malicious site**.

Host: evilsite.com

Django ensures that only trusted hostnames are allowed

ALLOWED_HOSTS = ['example.com']

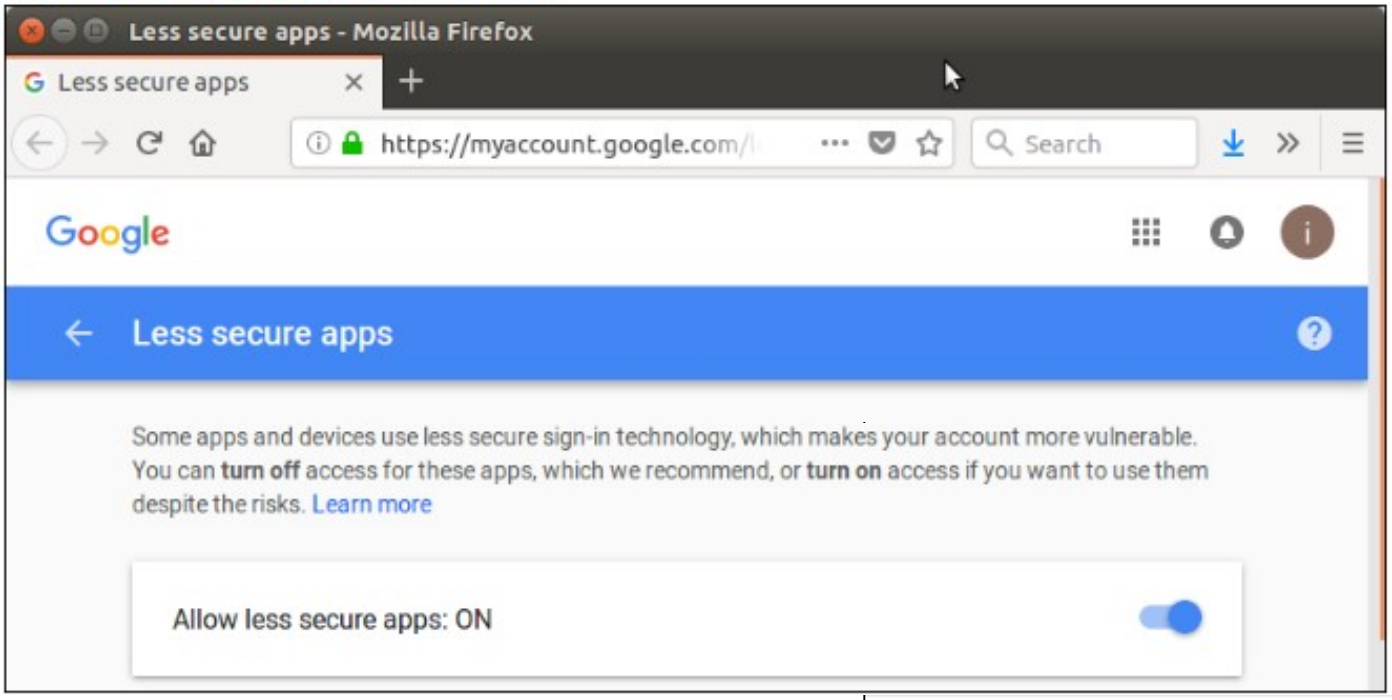
Django validates Host headers against the **ALLOWED_HOSTS** setting in the **django.http.HttpRequest.get_host()** method.

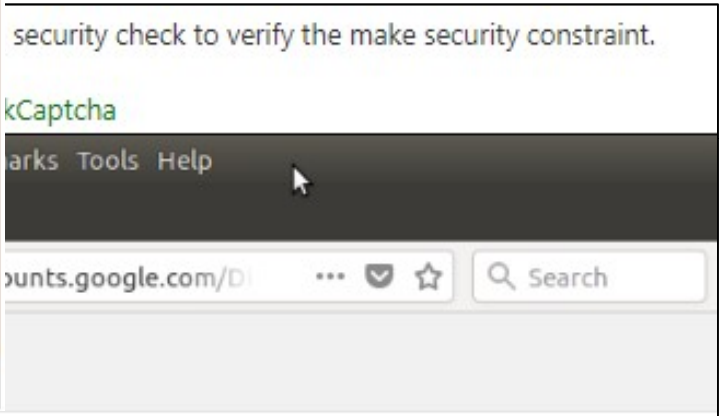
Configuring email & Sending emails with Django

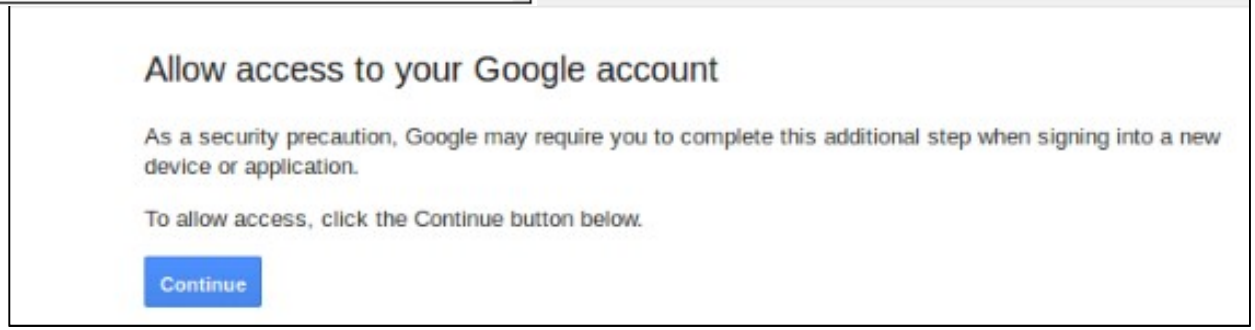
- Sending email using Django is pretty easy and require less configuration. In this lecture, we will send email to provided email.
- For this purpose, we will use **Google's SMTP** and a **Gmail account** to set sender.
- Django provides **built-in mail** library **django.core.mail** to send email.
- Before **sending** email, we need to make some changes in **Gmail account** because for security reasons **Google does not** allow direct access (login) by any application. So, login to the Gmail account and follow the urls.
- It will redirect to the **Gmail account settings** where we need to allow less secure apps but toggle the button. See the below screenshot.

Configuring email & Sending emails with Django

https://myaccount.google.com/lesssecureapps







Configuring email & Sending emails with Django

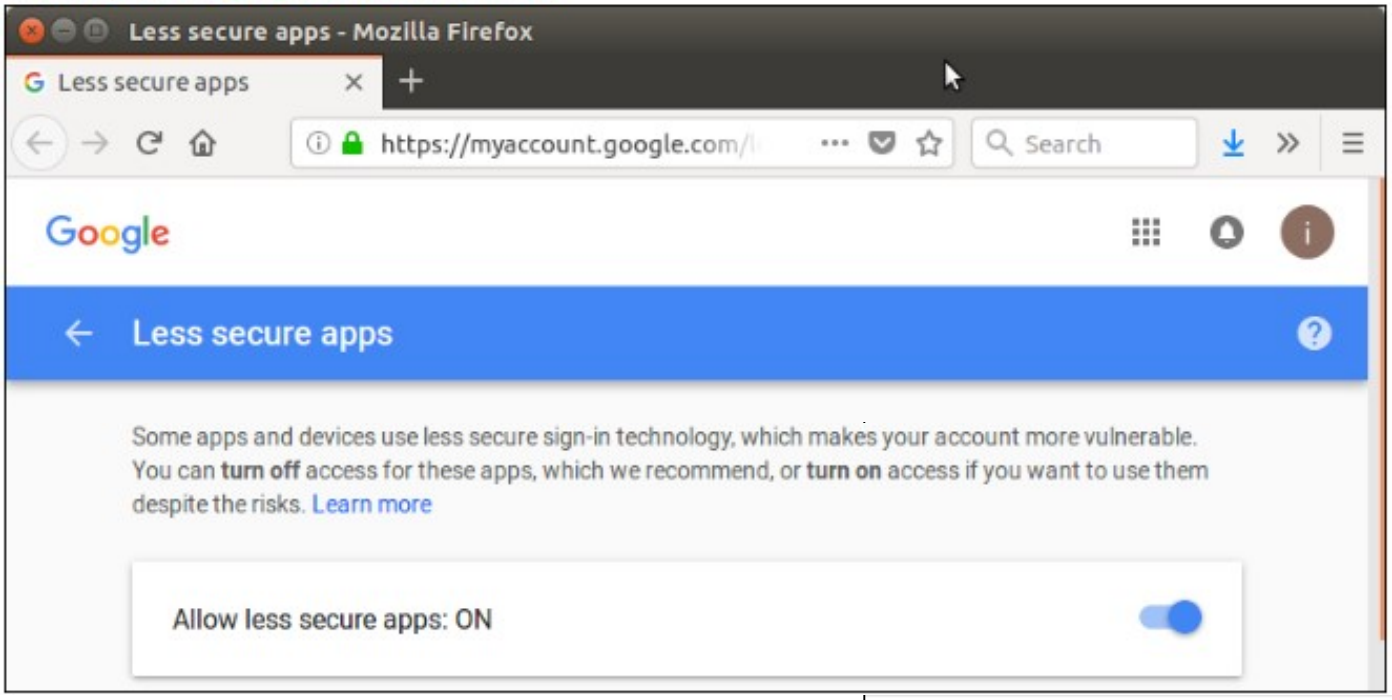
In django first we configure mail setup in setting.py

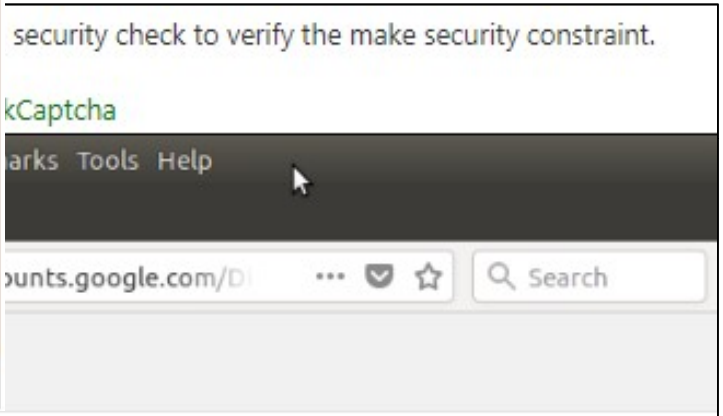
1. First make apps password from your gmail account from which you want to send mail
2. Now configure email setup in settings.py
3. Configure email setup

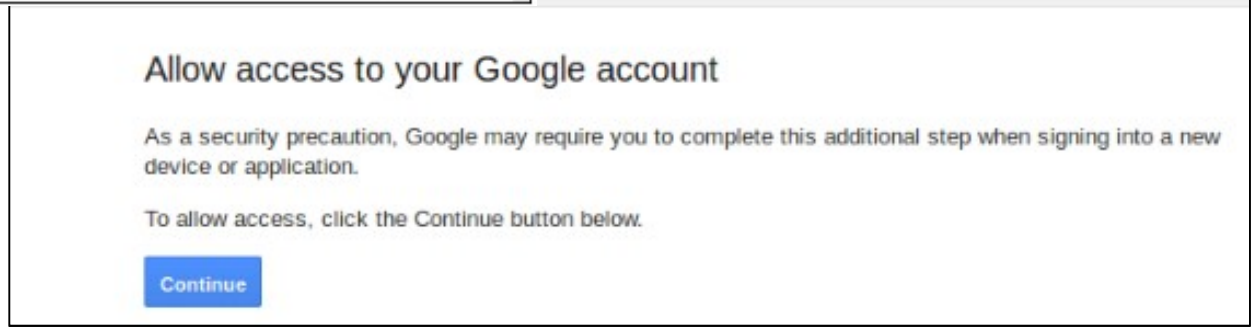
```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = ‘testmail@gmail.com'
EMAIL_HOST_PASSWORD =‘asdf hrhr qljh flvf'
EMAIL_USE_TLS= True
EMAIL_USE_SSL= False
```

Configuring email & Sending emails with Django

https://myaccount.google.com/lesssecureapps







Configuring email & Sending emails with Django

Sending email through Template

1. Make a file in your app folder **forms.py**

```
from django import forms class EmailForm(forms.Form):
name = forms.CharField(max_length=100)
subject = forms.CharField(max_length=100)
message = forms.CharField(widget=forms.Textarea)
recipient_email = forms.EmailField(label="Recipient Email")
```

2. **Create** function in **View** to handle input form value in your **app folder**

```
from django.shortcuts import render
from django.core.mail import send_mail
from .forms import EmailForm
from django.conf import settings
```

steps 2 in continue in next slide

Configuring email & Sending emails with Django

```
def send_email(request):
    if request.method == 'POST':
        form = EmailForm(request.POST)
        if form.is_valid():
                name = form.cleaned_data['name']
                subject = form.cleaned_data['subject']
                message = form.cleaned_data['message']
                recipient_email = form.cleaned_data['recipient_email']
                # Prepare the email body
                email_body = f"Message from {name}:\n\n{message}"
                # Send email
                send_mail( subject, email_body, settings.EMAIL_HOST_USER,
                [recipient_email], fail_silently=False, )
            return render(request, 'app/success.html')
    else:
        form = EmailForm() return
        render(request, 'app/send_email.html', {'form': form})
```

Configuring email & Sending emails with Django

3. Call form to you html file

```
<body> <h2>Send an Email</h2>
<form method="post">
        {% csrf_token %} {{ form.as_p }}
 <button type="submit">Send</button>
</form>
```

4. Make another html template to redirect after mail successfully

```
<body> <h2>Email sent successfully!</h2>
<a href="{% url 'send_email' %}">Send another</a>
```

5. Now update view file

Sending emails with Django

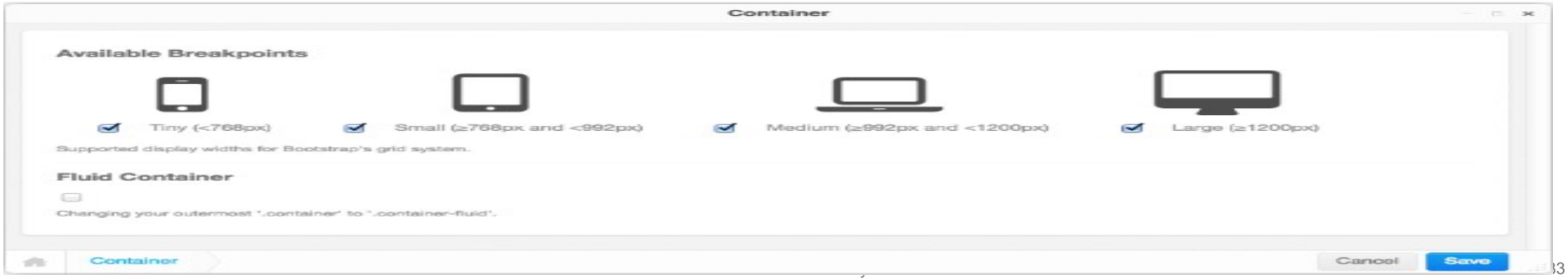
5. Now update view file

Adding Grid Layout On Registration Page

Bootstrap Container

- A Container is the outermost component the Bootstrap framework knows of. Here the designer can specify the breakpoints of a web page. **By default, Bootstrap offers 4 breakpoints: "large", "medium", "small" and "tiny".** These determine for which kind of screen widths, the grid system may switch the layout.

- The editor window for a Container element offers the possibility to deactivate certain breakpoints. While this might make sense under certain conditions, it is safe to always keep all four breakpoints active, since this gives the designer of the web page the maximum flexibility.



Adding Grid Layout On Registration Page

Small devices exclusively

If the web page shall be optimized just for small but not for large devices, then disable the breakpoints for **Large** and/or **Medium**. In the project's style-sheets, the maximum width of the container element then must be reduced to that chosen breakpoint:

```
@media(min-width: 1200px) {
  .container {
    max-width: 970px;
  }
}
```

or, if you prefers the SASS syntax:

```
@media(min-width: $screen-lg) {
  .container {
    max-width: $container-desktop;
  }
}
```

NIET Greater Noida Autonomous Institute

Adding Page Restrictions

Django provides a robust built-in authentication system. It **handles** user **accounts**, **groups**, **permissions**, and **cookie-based user sessions**.

This system includes:

Users: People **registered** with your website

Permissions: **Binary (yes/no) flags** designating whether a user may perform a certain task

Groups: A way of **categorizing users** to apply permissions to multiple users at once

Middleware and **Request Processing**

Django uses **middleware** to process each request before it reaches the view. The authentication middleware adds the **request.user object** to every request. This **object represents** the current user and is how Django keeps **track** of **authenticated** users.

Adding Page Restrictions

Decorators

Decorators are a **key concept in Python** used **extensively** in Django.

They **modify** the **behavior** of a function or class **without directly** changing its **source code**.

In the context of page restrictions, **decorators wrap** view functions to add security checks.

The **@login_required** Decorator

This decorator **checks** if the **current user is authenticated**. If not, it **redirects** to the login It works by:

a) Checking the **request.user.is_authenticated** property (b) If true, it **allows** the view to **process normally** (c) If false, it **redirects** to the login URL with the **current absolute path** in the query string

NIET Greater Noida Autonomous Institute

Adding Page Restrictions

The @permission_required Decorator

This decorator checks if the user has a specific permission. It works by:

(a) Ensuring the user is **authenticated**

(b) Checking if the user has the **specified permission** using the **user.has_perm()** method

(c) If the user **lacks** the **permission**, it either raises an **exception** or **redirects** to the login page

Login Functionality Test and Logout

- Exaplaining Django Login and Logout. Django is a High-Level Web Framework and it has lots of built-in features. We can use those built-in functions for our common use of Web Application. Some of the functions are Permission and User Control, Signals, Templates, Django ORM, Access Control List, etc. Out of this Registration App, is a good example and a good thing about it is that the features can be used out-of-the-box.

- With the Authentication Views, you can take advantage of the following features

1. Login
2. logout
3. User Registration
4. Change Password
5. Reset Password or Forgot Password

Login Functionality Test and Logout

Configure the Django Authentication URL Routes

In your urls.py file, import django.contrib.auth.views module and add the URLconf for Login and Logout.

```
from django.contrib import admin
from django.urls import path

from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),

    # Login and Logout
    path('login/', auth_views.LoginView.as_view(), name='login'),
    path('logout/', auth_views.LogoutView.as_view(), name='logout'),
]
```

Login Functionality Test and Logout

Create a Django Login Template & Form

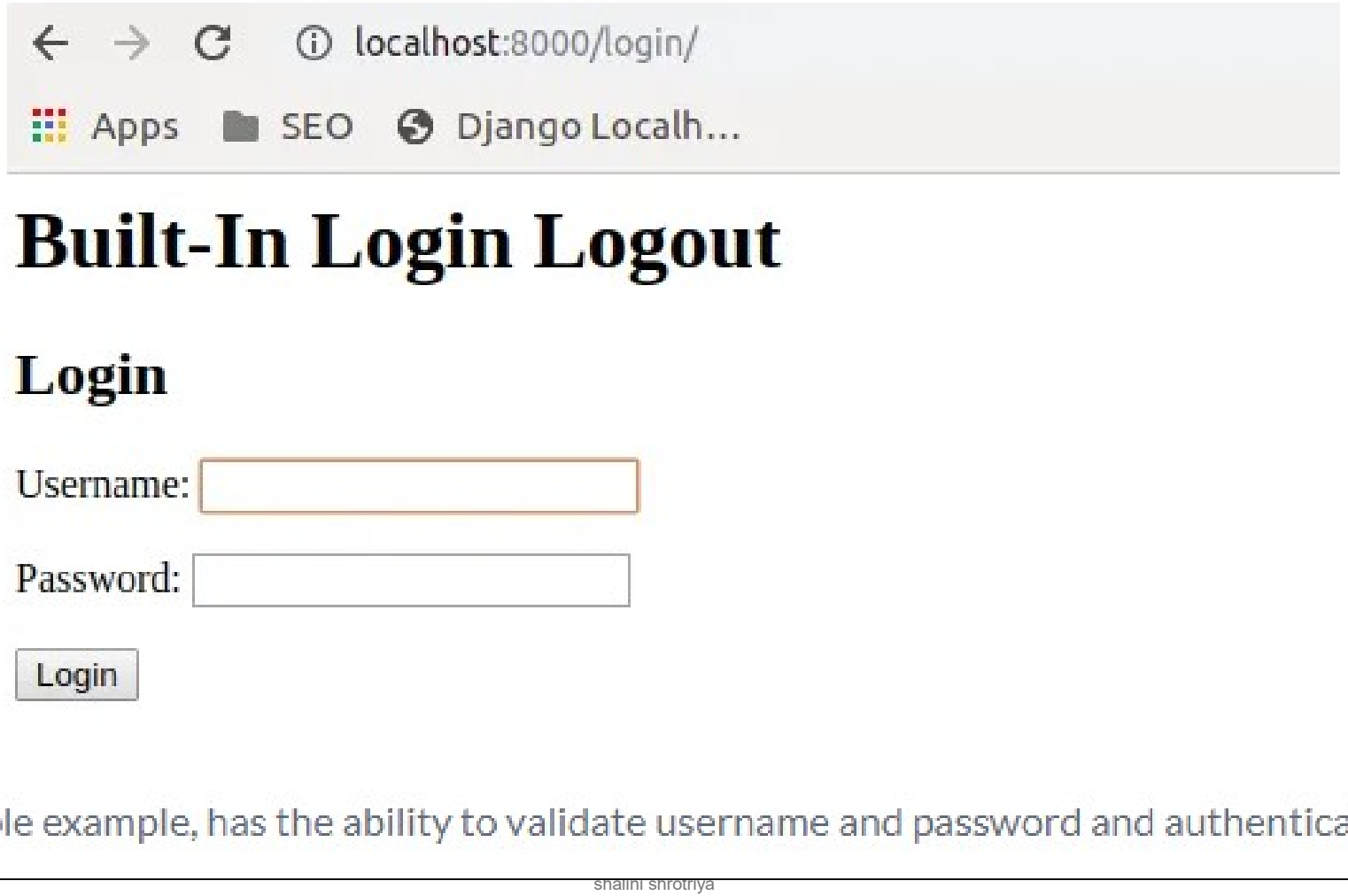
By default, **LoginView** will try to render `registration/login.html`. So in your `templates` folder, create a `registration` folder and inside that create `login.html` file.

```
{% extends 'base.html' %}

{% block title %}Login Page{% endblock title %}

{% block content %}
    <h2>Login</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Login</button>
    </form>
{% endblock content %}
```

Login Functionality Test and Logout



NIET Greater Noida
Get Future Ready Autonomous Institute

Daily Quiz

1. Discuss Django Authentication System.
2. What is the role of frameworks in python.
3. Discuss any three frameworks.
4. Discuss about , Security Problem & Solution with Django .
5. Discuss implementation rule of Falcon.
6. Discuss the Zappa framework.
7. Discuss the role of Dash.
8. Discuss the application area of cherryPy.
9. Discuss about the Request Http methods in Python.
10. Discuss about Flask application.

Topic Link (YouTube & NPTEL Video Links)

YouTube /other Video Links

- https://youtu.be/eoPsX7MKfe8?list=PLIdgECt554OVFKXRpo_kuI0XpUQKk0ycO
- https://youtu.be/tA42nHmmEKw?list=PLh2mXjKcTPSACrQxPM2_1Ojus5HX88ht7
- https://youtu.be/8ndsDXohLMQ?list=PLDsnL5pk7-N_9oy2RN4A65Z-PEnvtc7rf
- https://youtu.be/QXeEoD0pB3E?list=PLsyeobzWxl7poL9JTVyndKe62ieoN-MZ3
- https://youtu.be/9MmC_uGjBsM?list=PL3pGy4HtqwD02GVgM96-V0sq4_DSinqvf

MCQ s

1. What is a Django App? ADjango app is an extended package with base package is Django B. Django app is a python package with its own components. C. Both 1 & 2 Option D. All of the above	2. Django was introduced by A. Adrian Holovaty B. Bill Gates C. Rasmus Lerdorf D. Tim Berners-Lee
3. What are Migrations in Django A. They are files saved in migrations directory. B. They are created when you run make migrations command. C. Migrations are files where Django stores changes to your models. D. All of the above	4. Which architectural pattern does django follow APHP B. MVT C. HTML D. None of the above

Glossary Questions

Top 10 Django interview questions

1. Explain Django Architecture?
2. Explain the Django project directory structure?
3. What are models in Django?
4. What are templates in Django or Django template language?
5. What are views in Django?
6. What is Django ORM?
7. What is Django Rest Framework(DRF)?
8. What is the difference between a project and an app in Django?
9. What are different model inheritance styles in the Django?
10. What are Django Signals?

NIET Greater Noida
Get Future Ready Autonomous Institute

Summary

Till now we understand : The idea of this module Introduction to Django Authentication System To prove someone is who they say they are, they must provide a password when creating an account, and again at any time they want to authenticate themselves.

This should be familiar: you go through this kind of workflow any time you sign up for a service like Twitter or Netflix.

Django is widely lauded for its ease-of-use and pragmatic design, but like all software it is susceptible to its own share of critical vulnerabilities. Django's open source popularity means that default attack vectors are also widely known.

References

(1) Tom Aratyn, "Building Django 2.0 Web Applications: Create enterprise-grade, scalable Python web applications easily with Django 2.0", 2nd Edition 2018, Packt Publishing.

(2) Nigel George, "Build a website with Django", 1st Edition 2019, GNW Independent Publishing Edition.

(3) Ray Yao," Django in 8 Hours: For Beginners, Learn Coding Fast!, 2nd Edition 2020, Independently published Edition.

(4) Harry Percival, "Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript", 2nd Edition 2019, Kindle Edition.

NIET
Greater Noida
Autonomous Institute

THANK YOU