UNIT-3

Integrating Accounts & Authentication on Django

Topic: - Introduction to Django Authentication System

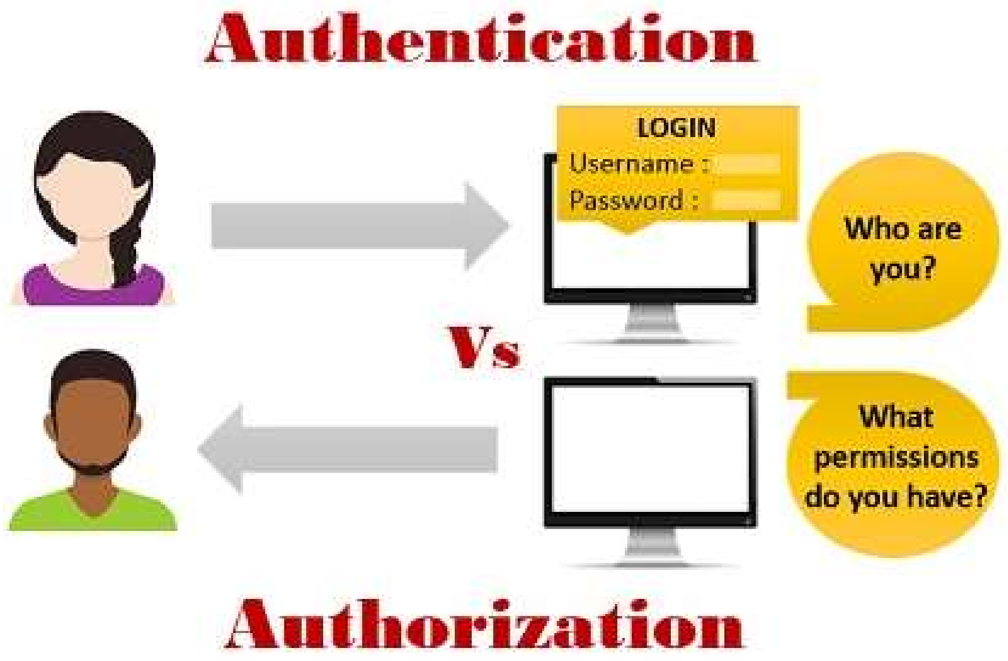
- Giving users the ability to create an account they can sign into is a common function for many websites.

- User might need an account to participate in a comment thread, save their personal information, or transfer money. Whatever the use case may be, you need to build an authentication system that's simple and safe for your users.

Difference between Authentication And Authorization:

- Usually, the term authentication refers to both the task, but there is a slight difference between these two terms.

- Authentication verifies a user who they claim to be, and authorization determines what an authenticated user is allowed to do.



Why Use Django User Authentication:

- It's better to use the existing built-in boilerplate than to reinvent the wheel.

- We will use the Django authentication system to create users who can sign up and comment on our blog website.
- The Django authentication system handles both authentication and authorization.

What has the Django authentication system got for us?

The Django Authentication System consists of :

1. Users
2. Permissions: Binary (yes/no) flags designation whether a user may perform a certain task.
3. Groups: A generic way of applying labels and permissions to more than one user.
4. A configurable password hashing system.
5. Forms and view tool for logging in users or restricting content.
6. A pluggable backend system.

Users :-

For most websites, the basic entity of authentication is a user. 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.

The primary attributes of the default user are:

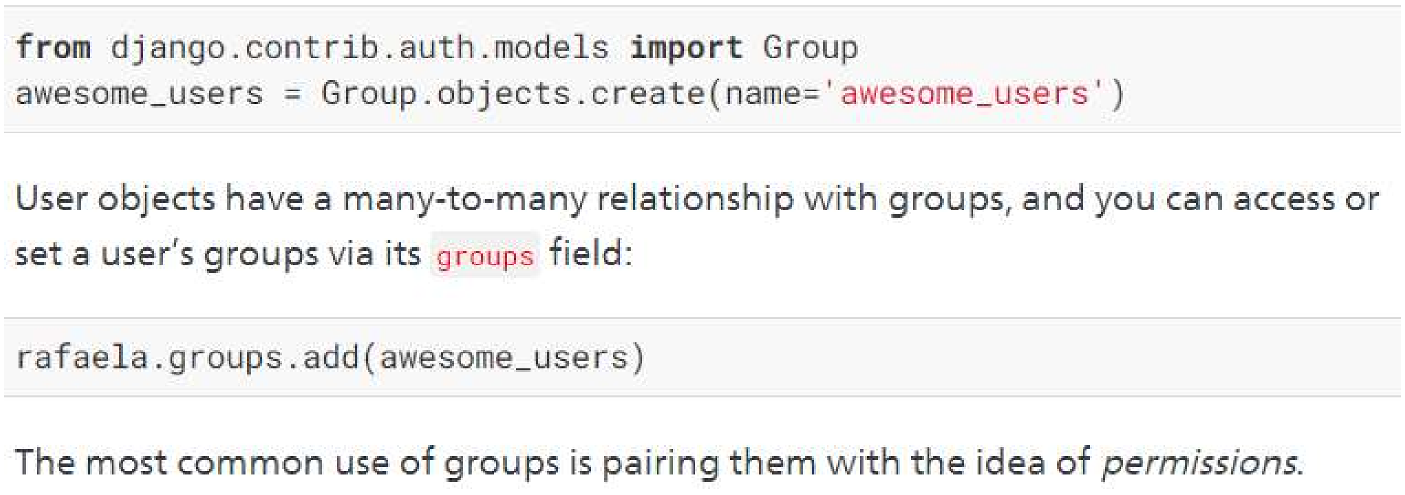
- username
- password
- email
- first_name
- last_name

```python
from django.contrib.auth.models import User
rafaela = User('rafaela', password='$uper$ecretpassword')
# OR
rafaela = User(
    'Rafaela',
    email='rafaela@example.com',
    password='$upser$ecretpassword',
    first_name='Rafaela',
    last_name='Lòpez',
)
```

Group:-

Django groups are, in short, a collection of users. Users can belong to multiple groups, but note that groups can't belong to other groups – so it's a shallow hierarchy. Groups are useful for creating "categories" of users for any number of things, like giving a particular group access to a feature on your website.

You can create a user group simply by giving it a name:



```
from django.contrib.auth.models import Group
awesome_users = Group.objects.create(name='awesome_users')
```

User objects have a many-to-many relationship with groups, and you can access or set a user's groups via its `groups` field:

```
rafaela.groups.add(awesome_users)
```

The most common use of groups is pairing them with the idea of *permissions*.

Permissions:-

- Django comes with a built-in permissions system. It provides a way to assign permissions to specific users and groups of users.

- It's used by the Django admin site, but you're welcome to use it in your own code.

The Django admin site uses permissions as follows and you can do like that :

Assuming you have an application with an **app_label foo** and a model named **Bar**, to test for basic permissions you should use:

- add: user.has_perm('foo.add_bar')
- change: user.has_perm('foo.change_bar')
- delete: user.has_perm('foo.delete_bar')
- view: user.has_perm('foo.view_bar')

The Permission model is rarely accessed directly.

User Authentication System

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

Django authentication provides both authentication and authorization together and is generally referred to as the authentication system.

By default, the required configuration is already included in the settings.py generated by django-admin startproject, these consist of two items listed in your INSTALLED_APPS setting:

'django.contrib.auth' contains the core of the authentication framework, and its default models.

'django.contrib.contenttypes' is the Django content type system, which allows permissions to be associated with models you create.

and these items in your MIDDLEWARE setting:

SessionMiddleware manages sessions across requests.

AuthenticationMiddleware associates users with requests using sessions.

django.contrib.auth

C:\Users\RK\AppData\Local\Programs\Python\Python38-32\Lib\site-packages\django\contrib\auth

models.py

class User

User Object - 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.

Topic: - Security Problem & Solution with Django

Security is super important the security of your application matters.

- Privacy matters.
- Trust matters.

Let’s look at three common acronyms that start with C and the problems they address.

- **CSRF** stands for Cross Site Request Forgery.
- **CORS** stands for Cross-Origin Resource Sharing.
- **CSP** stands for Content Security Policy

Django’s Top 10 Vulnerabilities:-

10. Session Modification:-

When session details are stored in the cache, root namespacing is used for both session identifiers and application-data keys. This can allow remote attackers to modify a session by triggering use of a key that is equal to that session's identifier.

9. Session Hijacking:-

Session hijacking involves an attacker gaining unauthorized access to a system using another user's session data. In this case, when using contrib.auth.backends.RemoteUserBackend, remote authenticated users can hijack web sessions via vectors related to the REMOTE_USER header.

8. Cache Poisoning:-

Cache poisoning occurs when incorrect data is inserted into a DNS resolver's cache, causing the name server to provide an incorrect IP address or destination. These versions of Django do not not properly include the:

- Vary: Cookie
- Cache-Control header in response

This can allow remote attackers to obtain sensitive information or poison the cache via a request from certain browsers.

7. Arbitrary URLs Generation:-

In these versions, the django.http.HttpRequest.get_host function allows remote attackers to generate and display arbitrary URLs via crafted username and password Host header values.

6. CSRF: Unauthenticated Forged Requests:-

CSRF is short for Cross Site Request Forgery, an attack that utilizes the user's web browser to perform an unwanted action on another website in which the user is currently signed in. The CSRF protection mechanism in these versions of Django do not properly handle web-server configurations supporting arbitrary HTTP Host headers, allowing remote attackers to trigger unauthenticated forged requests via vectors involving a DNS CNAME record and a web page containing JavaScript code.

5. CSRF via Forged AJAX Requests:-

These versions of Django do not properly validate HTTP requests that contain an X-Requested-With header, making it trivial for remote attackers to carry out cross-site request forgery (CSRF) attacks via forged AJAX requests that leverage a "combination of browser plugins and redirects to the page.

4. Directory Traversal:-

In these versions of Django, remote attackers are able to read or execute files via a / (slash) character in a key in a session cookie, related to session replays.

3. DoS: Via Unspecified Vectors:-

DoS is short for Denial of Service, and occurs when an attacker brings down a network/website by flooding it with data packets. The validators.URLValidator in these versions of Django allow remote attackers to cause a denial of service (CPU consumption) via unspecified vectors.

2. DoS: Via Multiple Requests with Unique Session Keys:-

The session backends in Django allows remote attackers to cause a denial of service (session store consumption) via multiple requests with unique session keys.

1. Type Conversion Vulnerability:-

In these versions of Django, the following field classes do not properly perform type conversion :

- FilePathField
- GenericIPAddressField
- IPAddressField

This gives remote attackers access to unspecified impact and vectors related to MySQL.

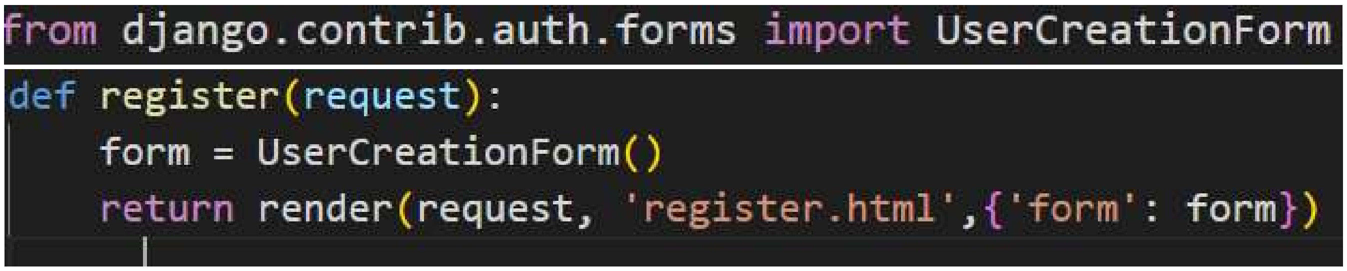
Remediation/ Solution:-

To fix the above vulnerabilities, you'll need to update the current working version of your Django framework in all your environments. And while Django is backwards compatible, it is nonetheless crucial that you identify any components in your web app that might be impacted by patching/updating.

Topic: - Creating Registration Form using Django

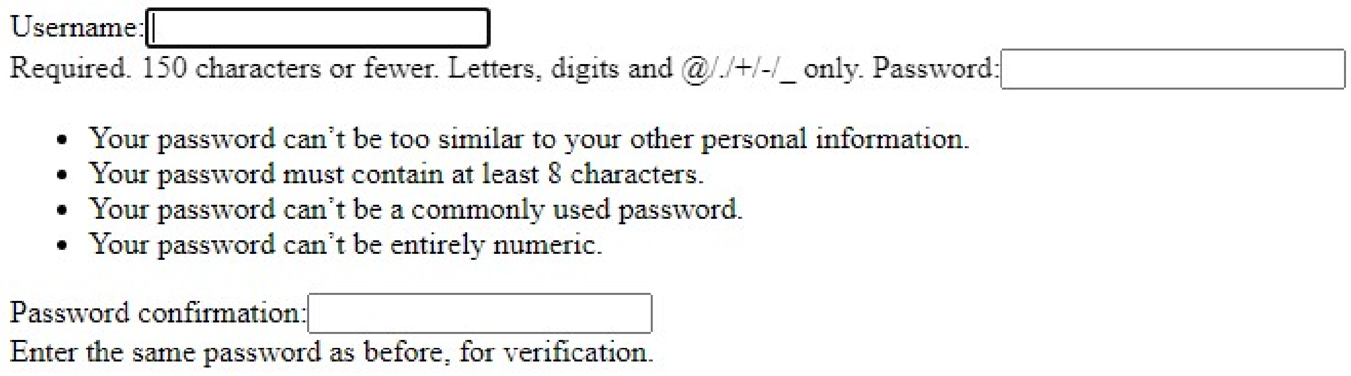
The steps of building a Django form (from the beginning) :-

- Step one — Create A Django Project
- Step two — Create A Django App insight the project
- Step three — Create a templates folder
- Step four — Create an index.html file
- Step five — Create a form_page.html file (optional)
- Step six — Create a forms.py file (optional)
- Step seven — Define index and form_name_view into the views.py file
- Step eight — Create a TEMPLATE_DIR in settings & assign it to the TEMPLATES list
- Step nine — Register your app into the INSTALLED_APPS list in the views.py
- Step ten — Import your app's views into the urls.py file of the project and map your URLs
- Step eleven — See if everything works & debug



```python
from django.contrib.auth.forms import UserCreationForm
```

```python
def register(request):
    form = UserCreationForm()
    return render(request, 'register.html',{'form': form})
```


Username:
Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only. Password:

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation:
Enter the same password as before, for verification.

Topic: - Adding Email Field in Forms

- EmailField in Django Forms is a string field, for input of Emails. It is used for taking text inputs from the user. The default widget for this input is EmailInput. It uses MaxLengthValidator and MinLengthValidator if max_length and min_length are provided. Otherwise, all inputs are valid.

Syntax:

field_name = forms.EmailField(**options)

Django form Email Field Explanation :-

1. **Enter the following code into forms.py file of my app.**

```
 from django import forms
 # creating a form
  class My Form(forms.Form):
  Example_field = forms.EmailField(max_length = 200)
```

2. **Add the my app to INSTALLED_APPS**

```
 # Application definition

INSTALLED_APPS = [
   'django.contrib.admin',
   'django.contrib.auth',
   'django.contrib.contenttypes',
   'django.contrib.sessions',
   'django.contrib.messages',
   'django.contrib.staticfiles',
   'my app',
]
```

3. **Now to render this form into a view we need a view and a URL mapped to that URL. Let's create a view first in views.py of myapp.**

```
from django.shortcuts import render
from .forms import MyForm

# Create your views here.
def home_view(request):
   context = {}
   context['form'] = MyForm()
   return render( request, "home.html", context)
```

4. Let's create a form in home.html

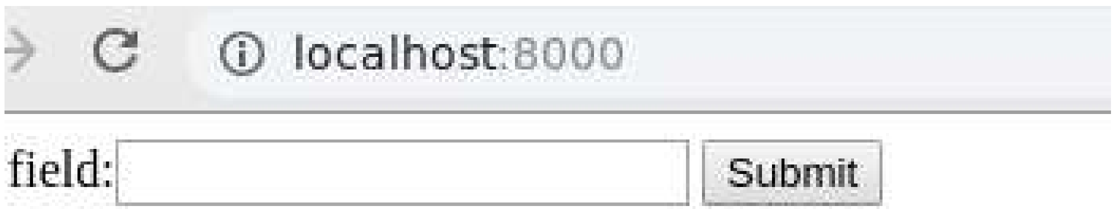
```
 <form method = "GET">

   {{ form }}

   <input type = "submit" value = "Submit">

</form>
```

5. Finally, a URL to map to this view in urls.py

```
from django.urls import path

# importing views from views..py

from .views import home_view



urlpatterns = [

   path('', home_view ),  ]
```

6. **Let's run the server and check what has actually happened, Run**

Python manage.py runserver



Topic: - Configuring email settings, sending emails with Django

Django
Send Email
In this article, we will learn to send emails to users automatically via the Django mail library, an extension of the smtplib module of Python.
django

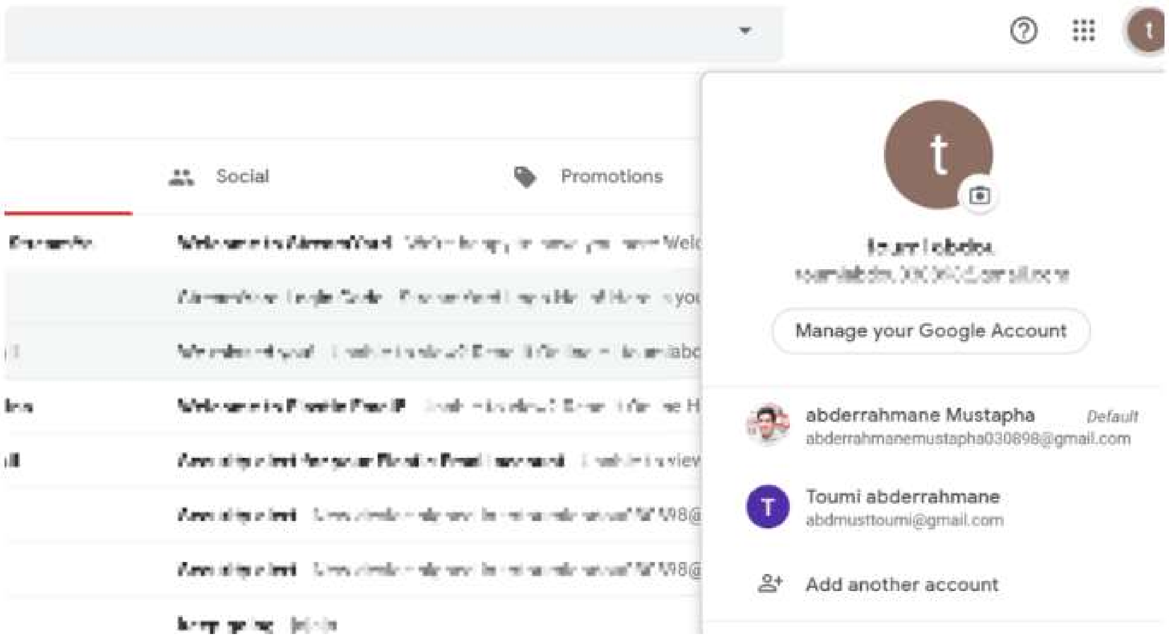
- Sending email using Django is pretty easy and require less configuration.

- For this purpose, we will use Google's SMTP and a Gmail account to set sender.

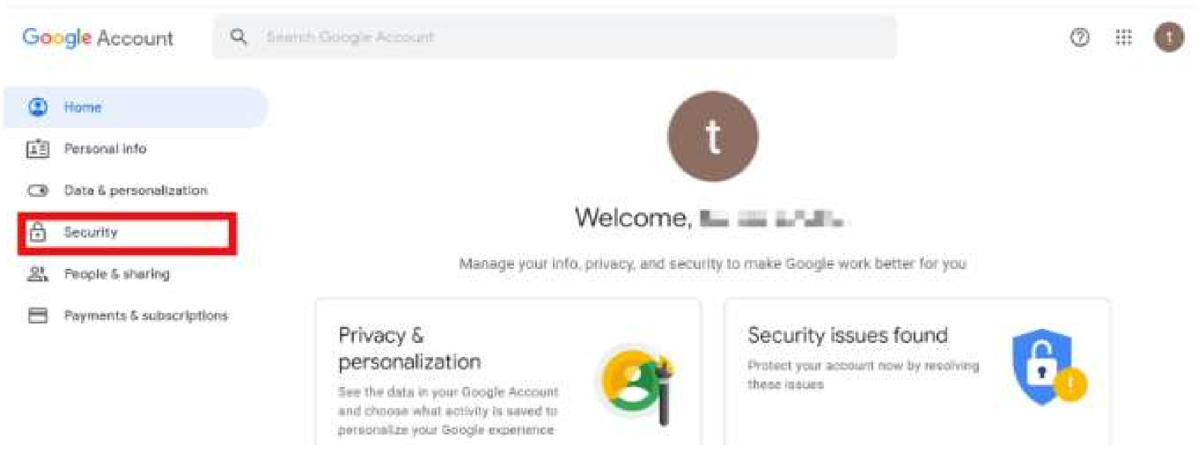
- Django provides built-in mail library **django.core.mail** to send email.

The Gmail part ✉

now you need to create a Gmail account and then click on **Manage your google account**



now click on the Security tab



Welcome, [redacted]

Manage your info, privacy, and security to make Google work better for you

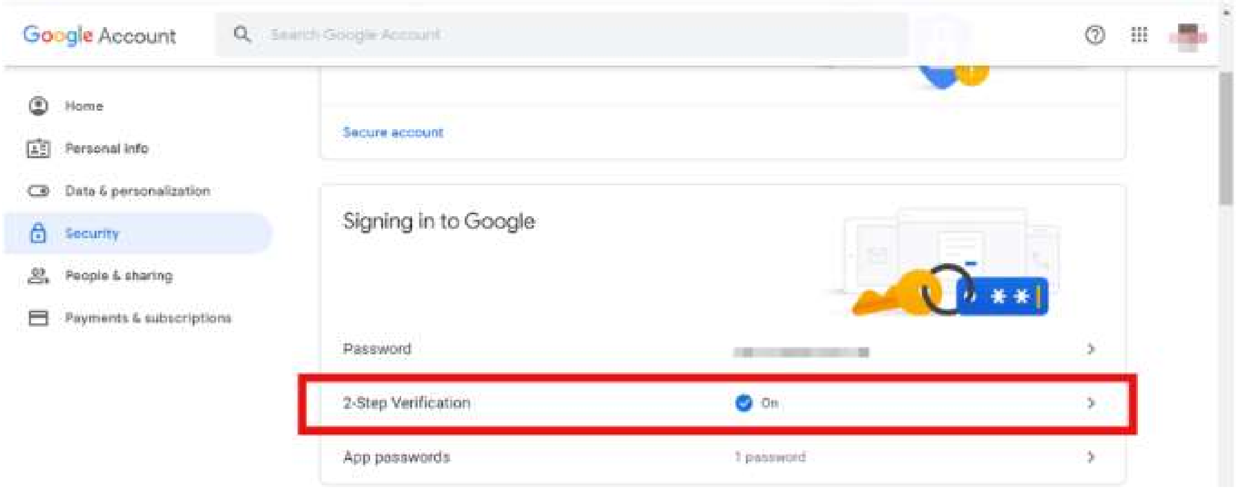
Privacy & personalization

See the data in your Google Account and choose what activity is saved to personalize your Google experience

Security issues found

Protect your account now by resolving these issues

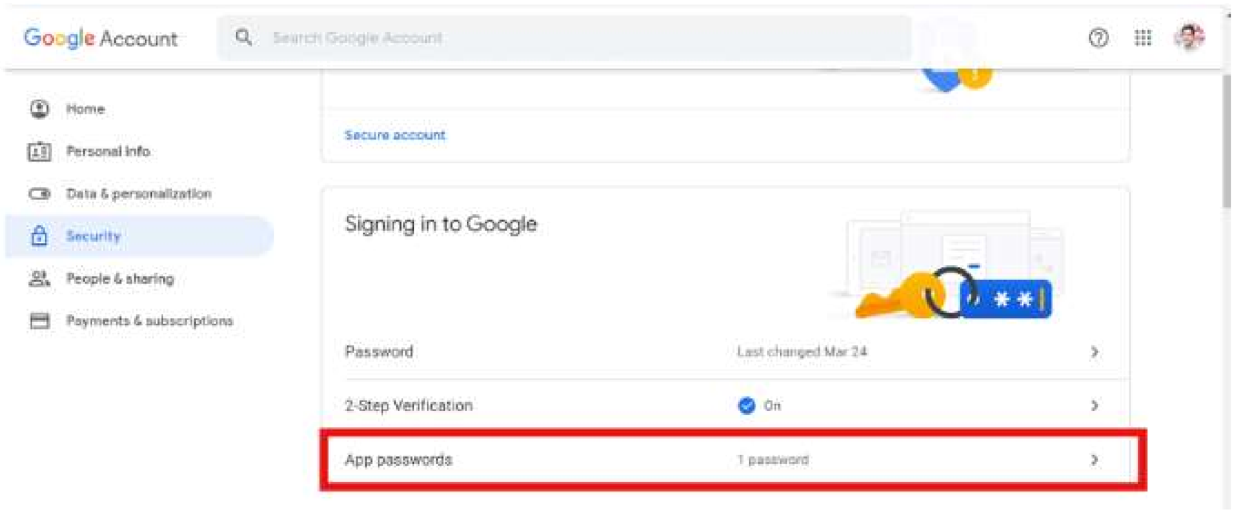
make sure to enable two steps verification



Signing in to Google

Password	[redacted]
2-Step Verification	On
App passwords	1 password

now click on App passwords



Google Account

Search Google Account

Home

Personal info

Data & personalization

Security

People & sharing

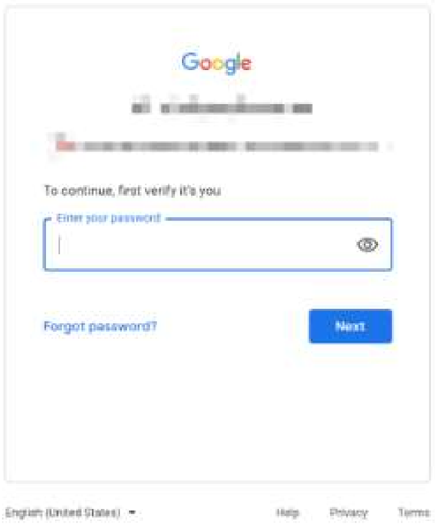
Payments & subscriptions

Secure account

Signing in to Google

Password	Last changed Mar 24	›
2-Step Verification	On	›
App passwords	1 password	›

you have to type your password again



Google

To continue, first verify it's you

Enter your password

Forgot password?

Next

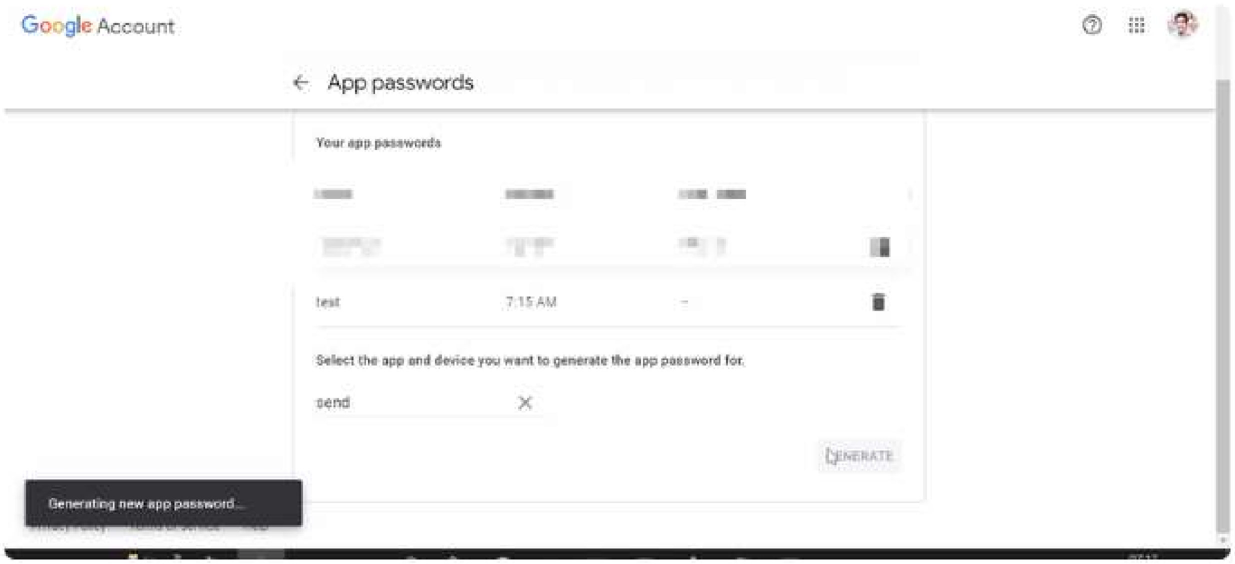
English (United States)

Help

Privacy

Terms

click on select app choose *** other (Custome Name) *** and give a name to you app



Google Account

App passwords

Your app passwords

test	7:15 AM	–	

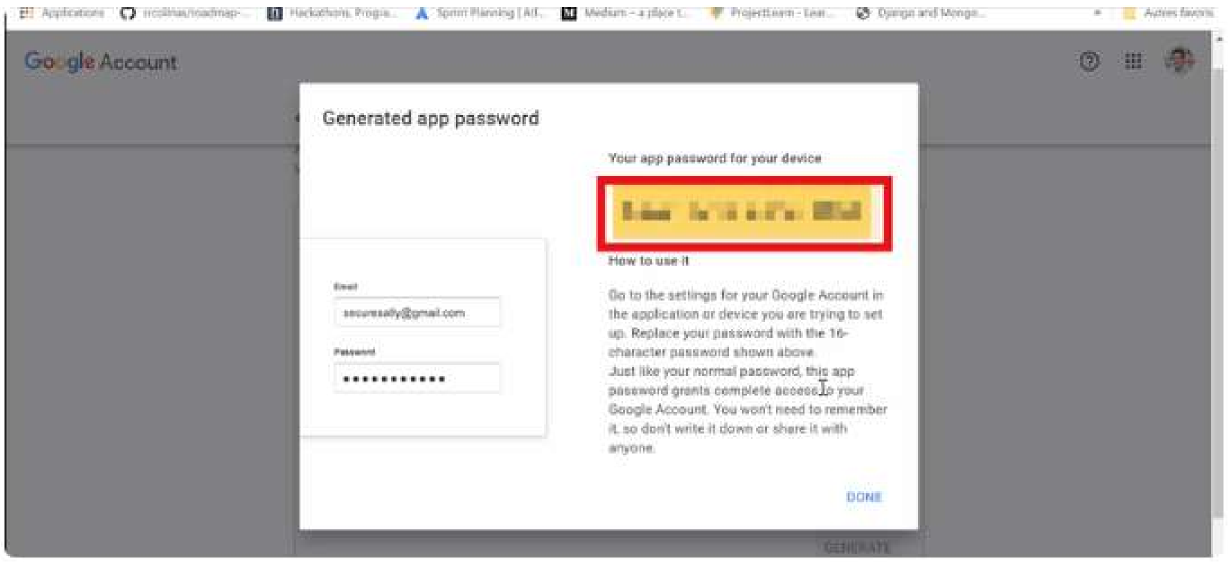
Select the app and device you want to generate the app password for.

send

GENERATE

Generating new app password...

the last step click on generate and Gmail will generate a key or an app password make sure to copy this key or save it in a text file



Google Account

Generated app password

Your app password for your device

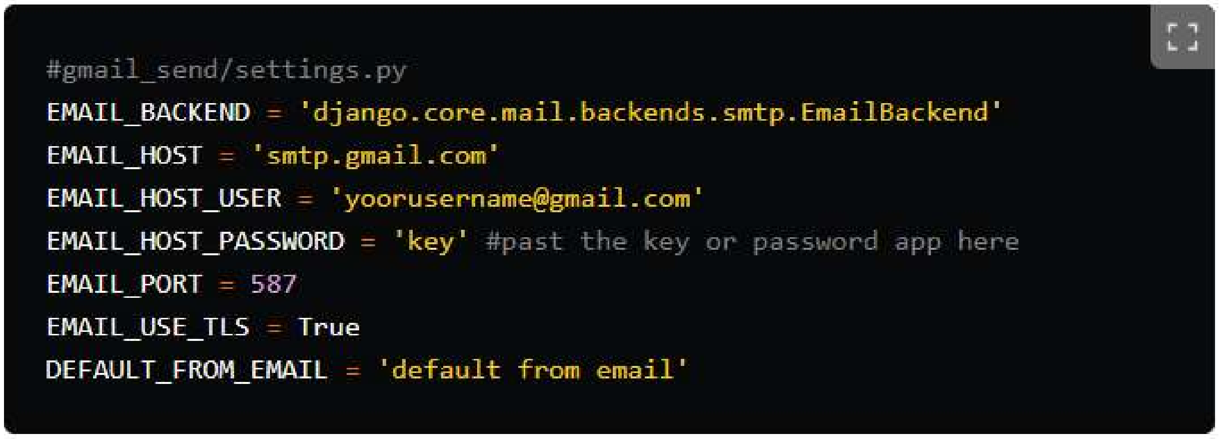
How to use it

Go to the settings for your Google Account in the application or device you are trying to set up. Replace your password with the 16-character password shown above.
Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.

Email: securesally@gmail.com
Password: ••••••••••••

DONE

Edit your settings.py file

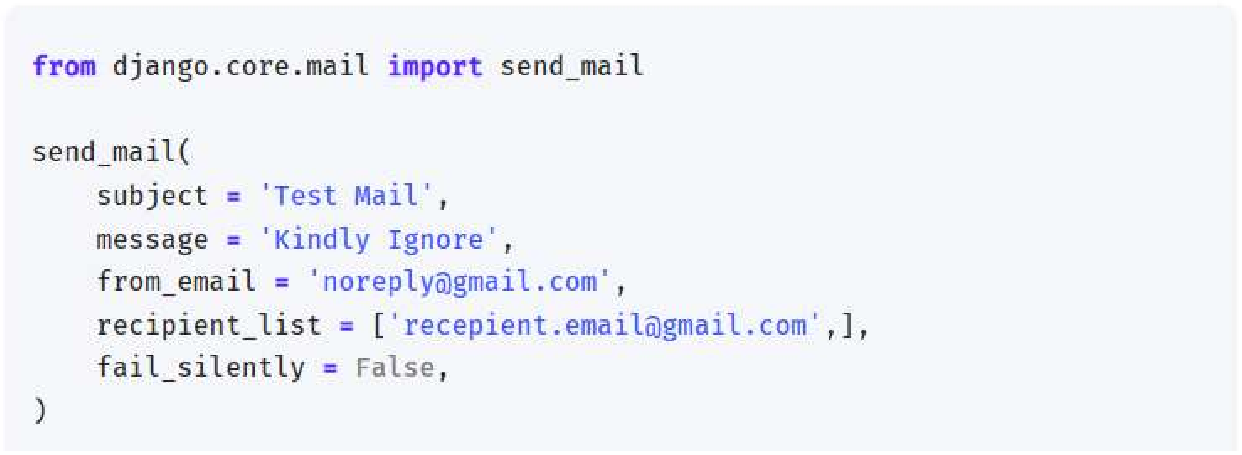


```
#gmail_send/settings.py
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = 'yoorusername@gmail.com'
EMAIL_HOST_PASSWORD = 'key' #past the key or password app here
EMAIL_PORT = 587
EMAIL_USE_TLS = True
DEFAULT_FROM_EMAIL = 'default from email'
```

<u>Writing Code to send Emails :-</u>

<u>1. Send emails individually using send_mail()</u>

In Views.py, add the code:



```
from django.core.mail import send_mail

send_mail(
    subject = 'Test Mail',
    message = 'Kindly Ignore',
    from_email = 'noreply@gmail.com',
    recipient_list = ['recepient.email@gmail.com',],
    fail_silently = False,
)
```

2. Sending Multiple Emails using send_mass_mail()

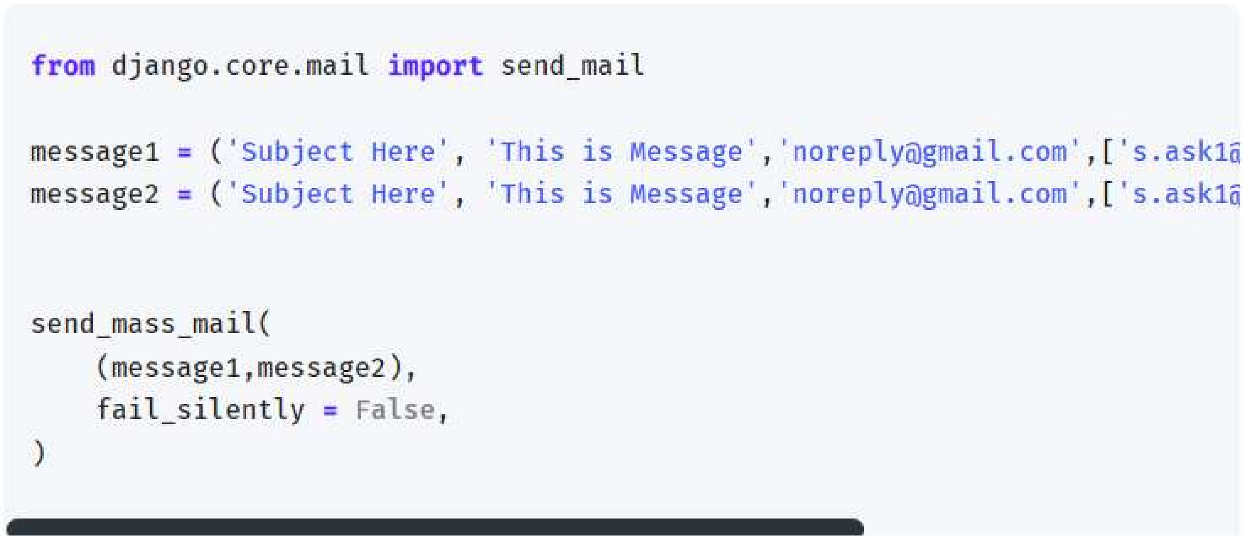
The steps to send out bulk emails. We'll use the send_mass_mail() method here. The syntax for send_mass_mails:

```
send_mass_mail(
    (datatuple),
    fail_silently = False,
)
```

Here the datatuple is the tuple containing the information about individual emails.

```
message1 = (subject, message, from_email, recipient_list)
message2 = (subject, message, from_email, recipient_list)
message3 = (subject, message, from_email, recipient_list)

send_mass_mail((message1,message2,message),fail_silently =False)
```

In views.py the code will look like:



```
from django.core.mail import send_mail

message1 = ('Subject Here', 'This is Message','noreply@gmail.com',['s.ask1@
message2 = ('Subject Here', 'This is Message','noreply@gmail.com',['s.ask1@


send_mass_mail(
    (message1,message2),
    fail_silently = False,
)
```

3. Send Emails using Django EmailMessage() Method :-

This method is used to send advanced mails, having features like BCC, CC, or attachments. This Django Method is handled by Email Backend.

The Email Backend class requires three steps:

1. **connection.open():** Ensures a long-term connection for sending mails.
2. **connection.close()**: Stops the established connection
3. **send_message():** Sends the mails; if the connection was not already open, then it temporarily opens the connection to send the mail.

The placeholder syntax is :

```
email1 = EmailMessage(
    subject = 'This is subject',
    message = 'This is message',
    from_email = 'from@gmail.com',
    to = ['mail1@gmail.com',],
    bcc = ['mail2@gmail.com'],
    cc = ['mail3@gmail.com'],
)
```

Therefore add the following code in view.py:

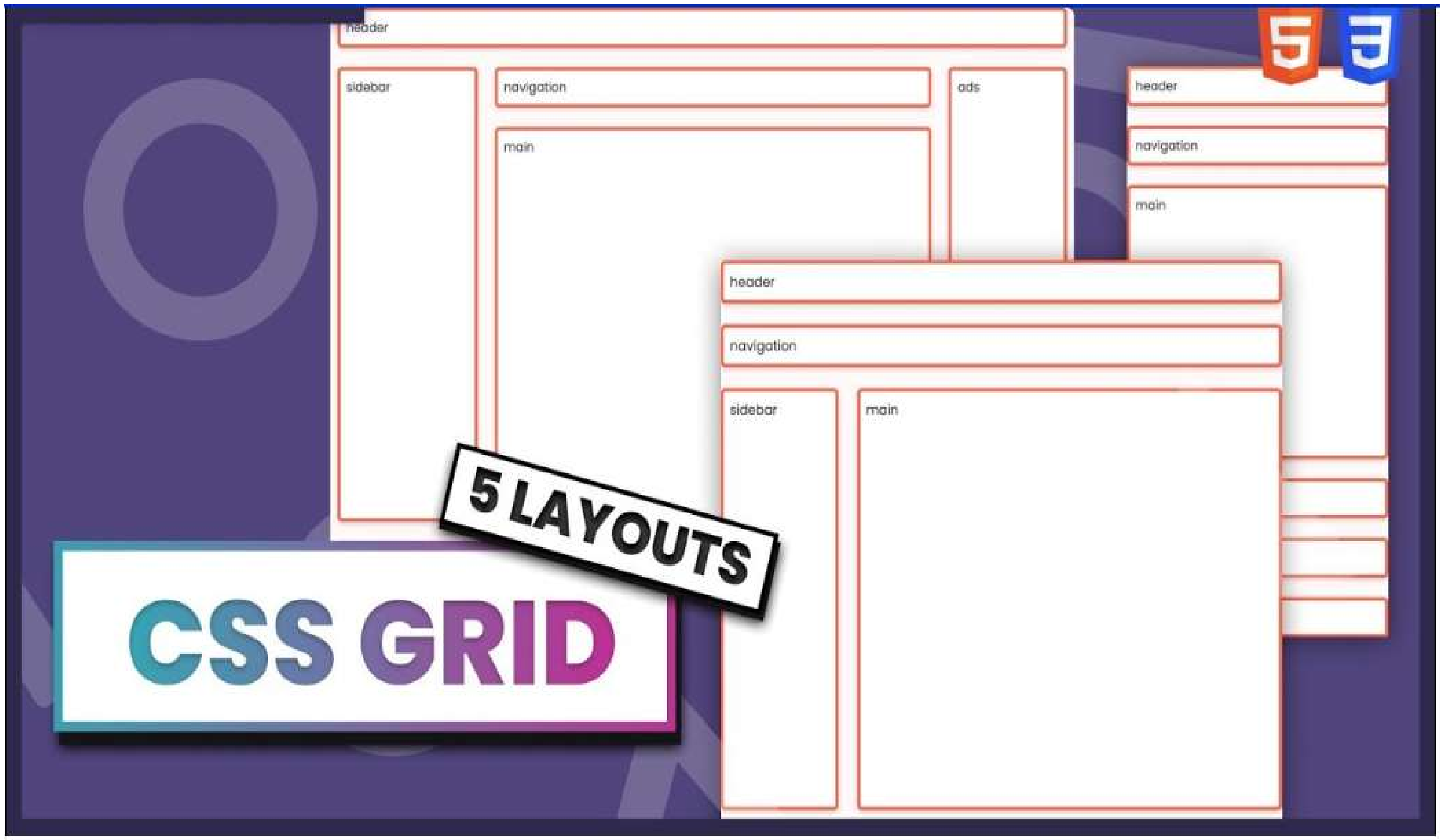
```
from django.core import mail
connection = mail.get.connection()

connection.open()

email = mail.EmailMessage(
    subject = 'Test Mail',
    message = 'Kindly Ignore',
    from_email = 'noreply@gmail.com',
    to = ['vn.ask1@gmail.com',],
    bcc = ['vn.ask2@gmail.com'],
)

connection.send_messages(email)
connection.close()
```

File Attachment:

EmailMessages() method provides **.attach_file('path to the file')** method to send attachments along with your email message. You can add the following code to attach the file:

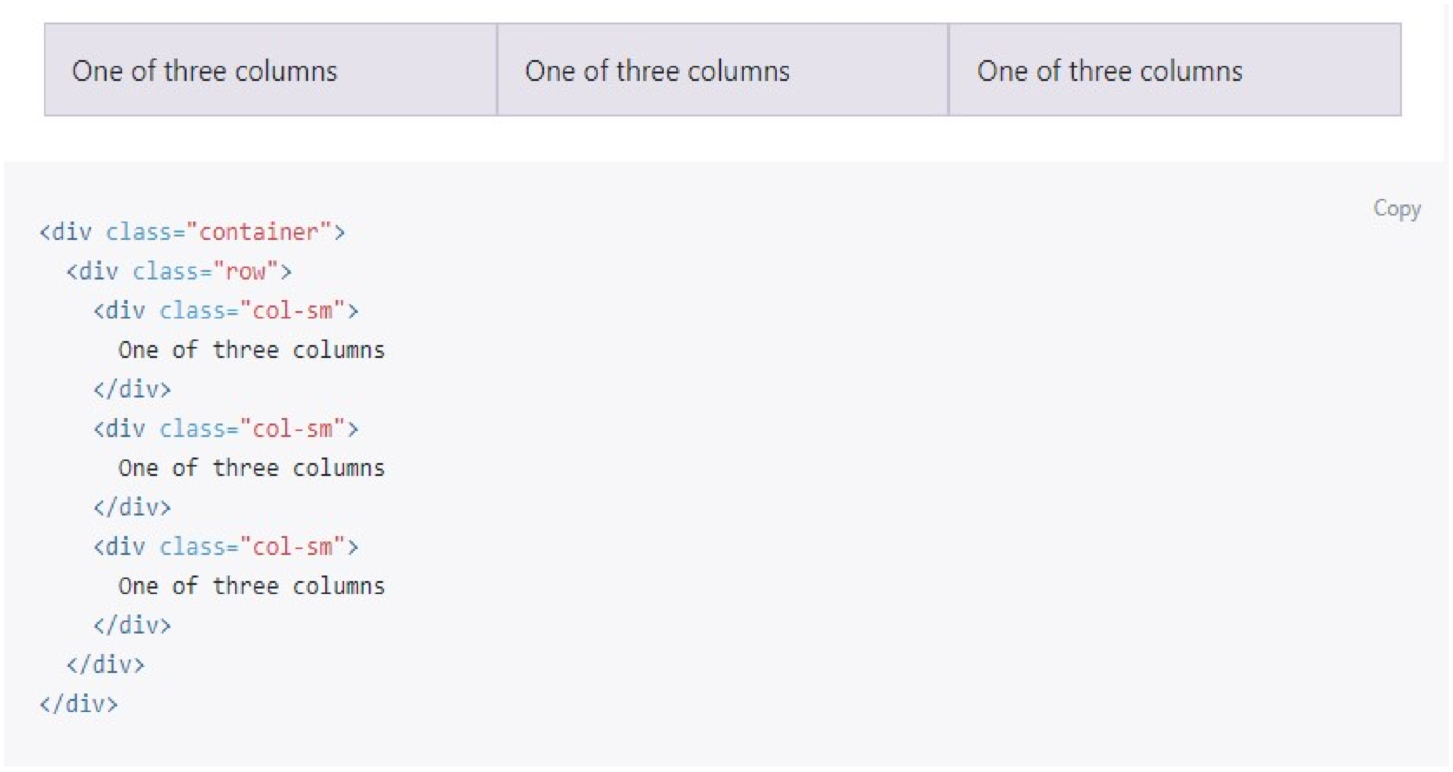
```
email1.attach_file('home/Desktop/books.jpg')
```

Topic: - Adding Grid Layout on Registration Page



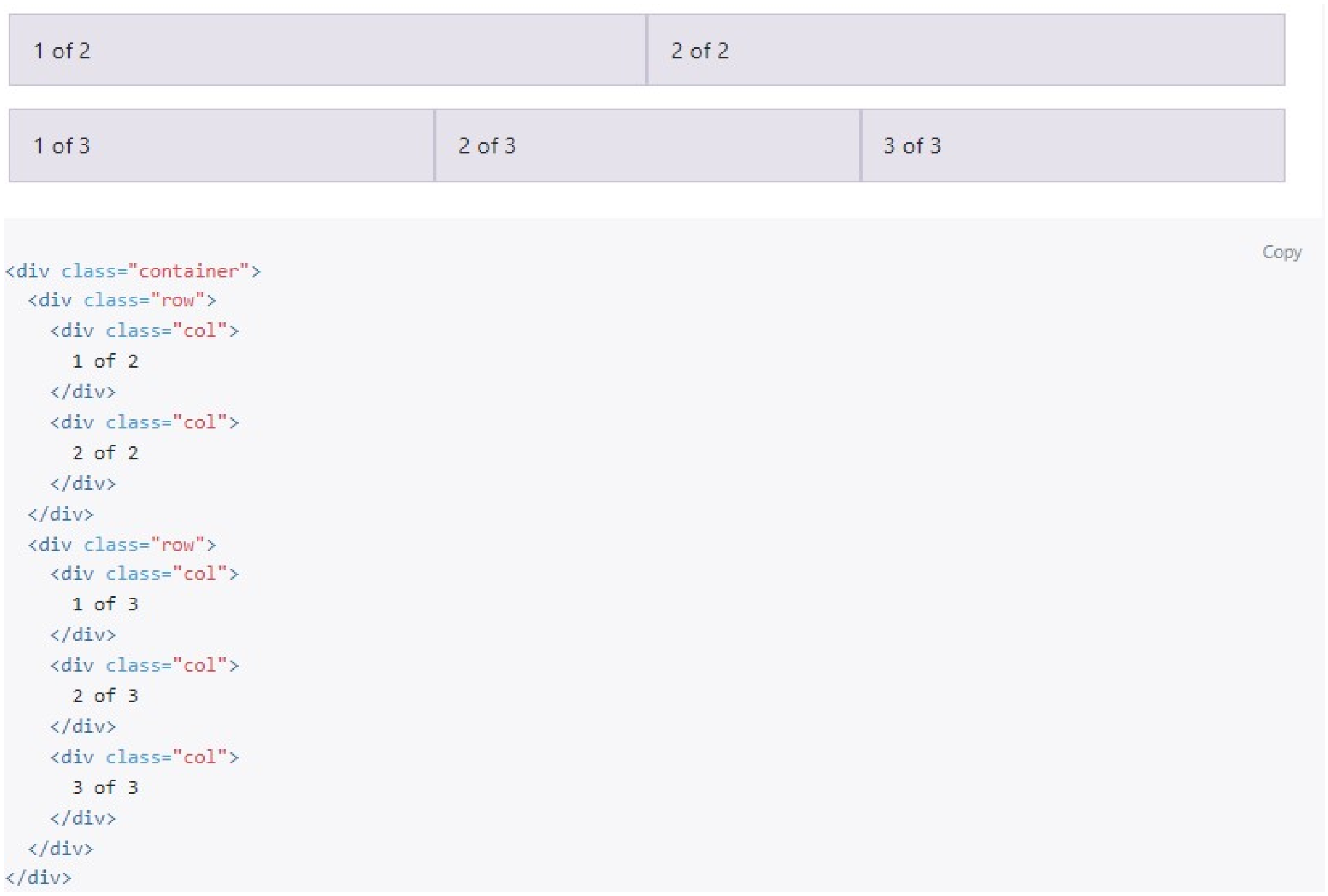
Grid system:-

Bootstrap’s grid system uses a series of containers, rows, and columns to layout and align content. It is fully responsive. Below is an example and an in-depth look at how the grid comes together.



```
<div class="container">
  <div class="row">
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
    <div class="col-sm">
      One of three columns
    </div>
  </div>
</div>
```

Example: - 2

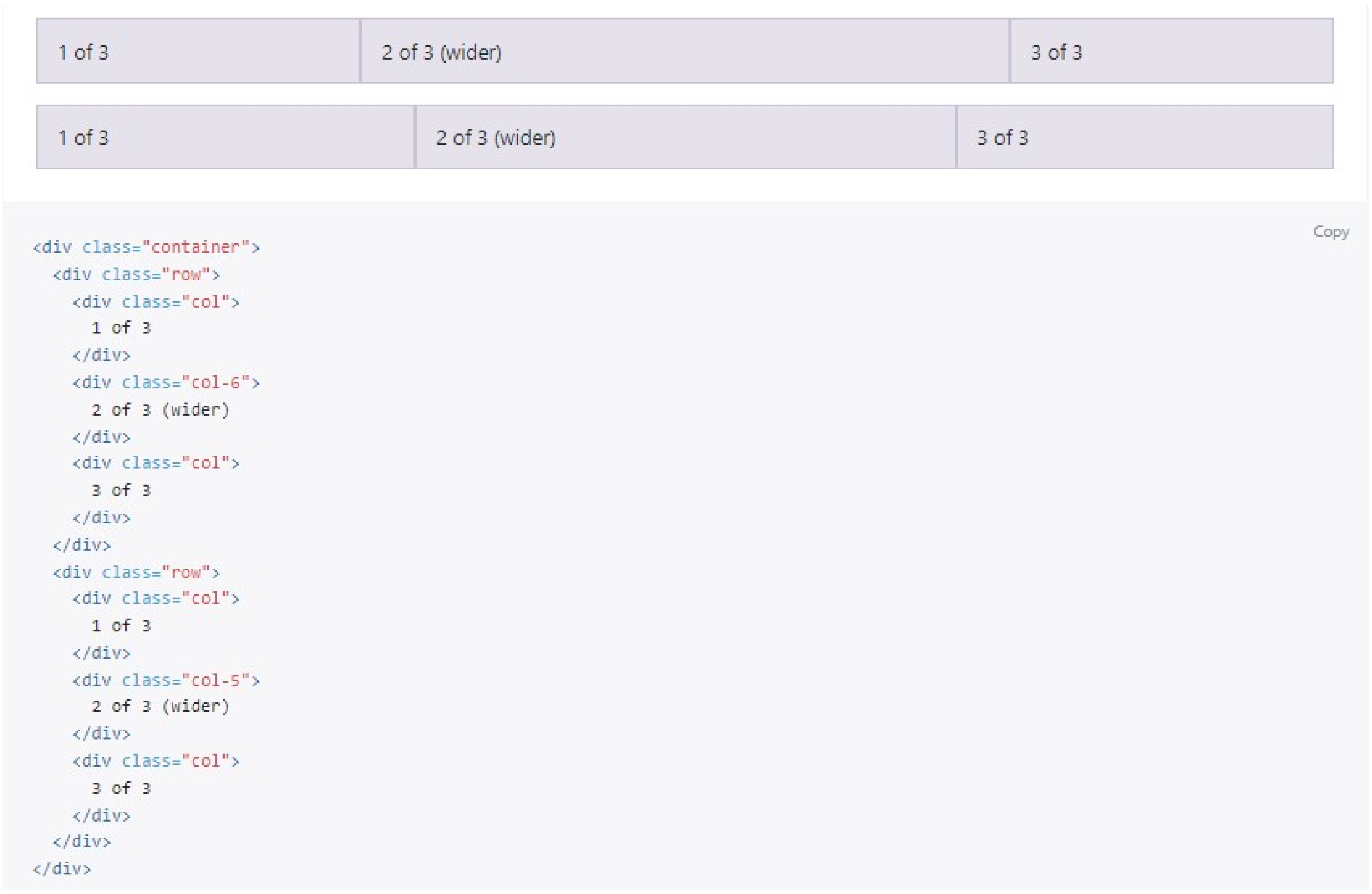


1 of 2 | 2 of 2

1 of 3 | 2 of 3 | 3 of 3

```html
<div class="container">
  <div class="row">
    <div class="col">
      1 of 2
    </div>
    <div class="col">
      2 of 2
    </div>
  </div>
  <div class="row">
    <div class="col">
      1 of 3
    </div>
    <div class="col">
      2 of 3
    </div>
    <div class="col">
      3 of 3
    </div>
  </div>
</div>
```

Example:- 3



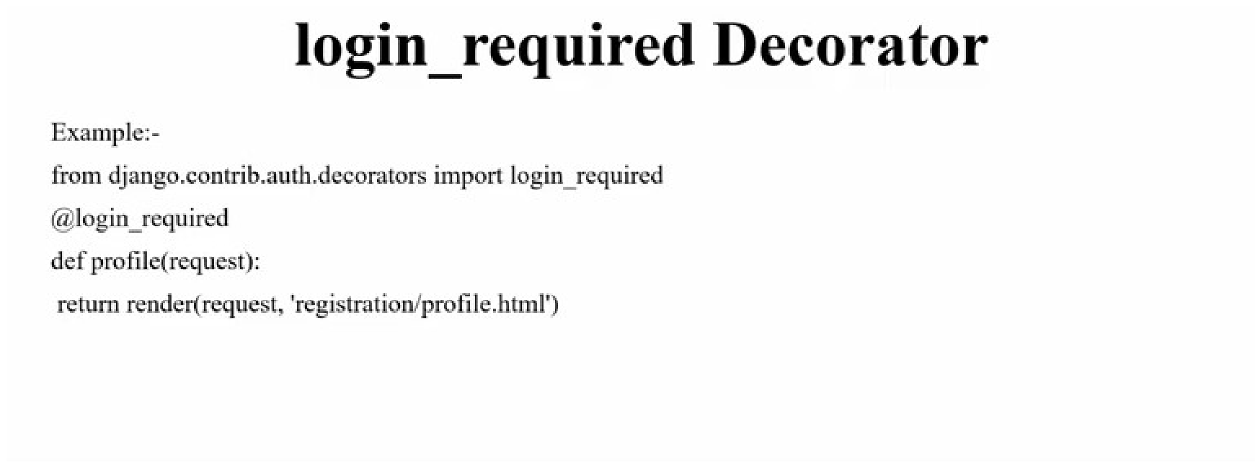
```
<div class="container">
  <div class="row">
    <div class="col">
      1 of 3
    </div>
    <div class="col-6">
      2 of 3 (wider)
    </div>
    <div class="col">
      3 of 3
    </div>
  </div>
  <div class="row">
    <div class="col">
      1 of 3
    </div>
    <div class="col-5">
      2 of 3 (wider)
    </div>
    <div class="col">
      3 of 3
    </div>
  </div>
</div>
```

Topic: - Adding Page Restrictions, Login & Logout.

How to restrict access with Django **required decorator function:-**

The Django **decorator** provide the feature to restrict the access. We have often visited websites in which we need to log in first before accessing or visiting other pages. In other words, restricting access.

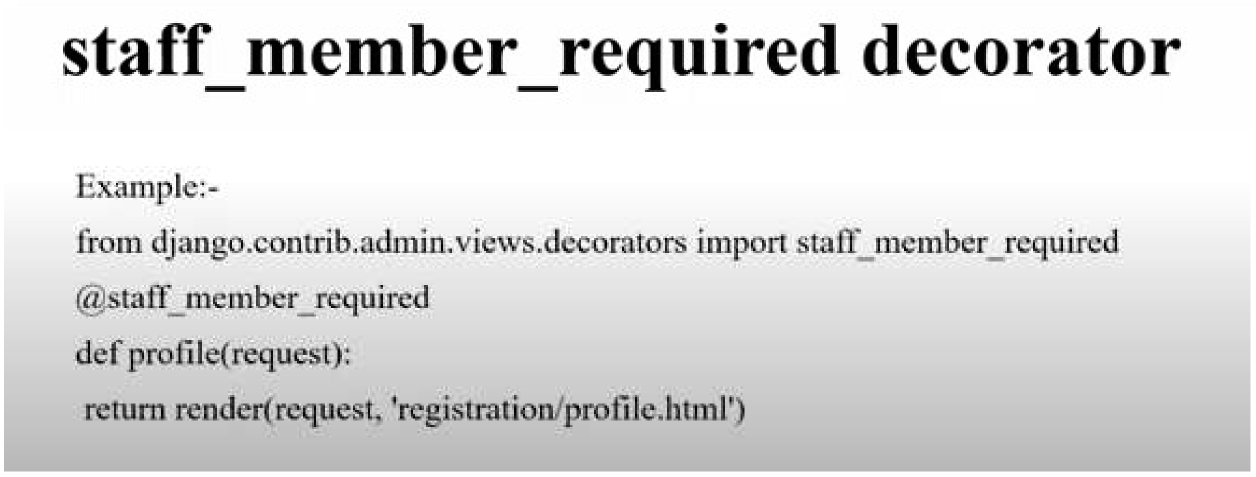
Django come with some built-in decorators, like login_required, require_POST or has_permission.



login_required Decorator

Example:-

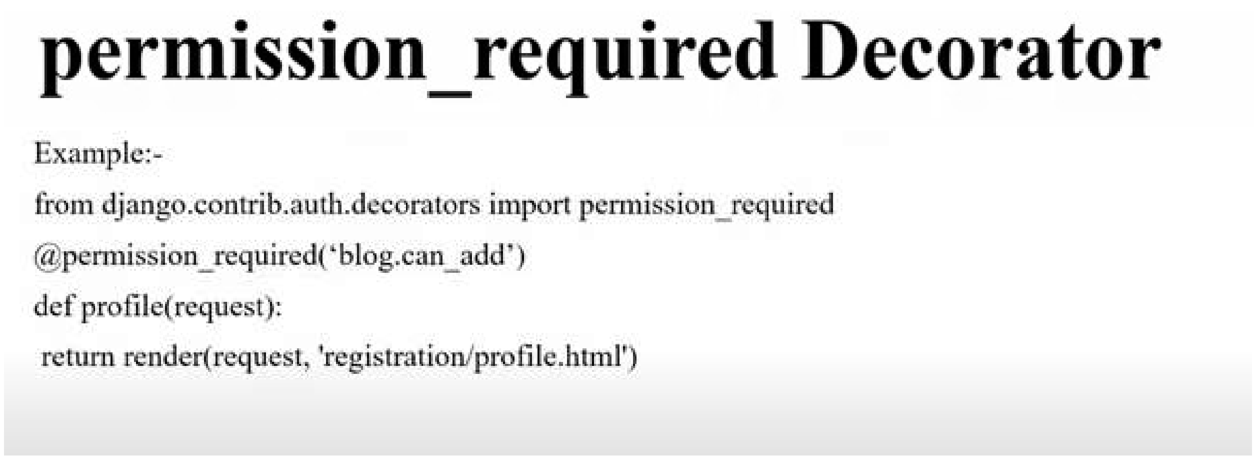
```
from django.contrib.auth.decorators import login_required
@login_required
def profile(request):
 return render(request, 'registration/profile.html')
```



staff_member_required decorator

Example:-

```
from django.contrib.admin.views.decorators import staff_member_required
@staff_member_required
def profile(request):
 return render(request, 'registration/profile.html')
```



permission_required Decorator

Example:-

```
from django.contrib.auth.decorators import permission_required
@permission_required('blog.can_add')
def profile(request):
 return render(request, 'registration/profile.html')
```