Course Module 1: Course Unit 11: Django (Advance Concepts)

Week 14

Django Intermediate Level Topics

Django Intermediate Level Topics

- Django Admin Interface
- Django Database
- Django Redirect
- Django Cookies Handling
- Django Form Handling and Validation
- Django File Upload
- Django Static File Handling
- Django Bootstrap
- Django CRUD

Django Advanced Topics

Django Advanced Topics

- **Django Sessions**
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Sessions

Cookies are small text files stored and maintained by the browser. It contains some information about the user and every time a request is made to the same server, the cookie is sent to the server so that the server can detect that the user has visited the site before or is a logged in user.

The cookies also have their drawbacks and a lot of times they become a path for the hackers and malicious websites to damage the target site.

Django Sessions

Since cookies store locally, the browser gives control to the user to accept or decline cookies. Many websites also provide a prompt to users regarding the same.

Cookies are plain text files, and those cookies which are not sent over HTTPS can be easily caught by attackers. Therefore, it can be dangerous for both the site and the user to store essential data in cookies and returning the same again and again in plain text.

Django Sessions

The **session** is a semi-permanent and two-way communication between the server and the browser.

- **semi** means that session will exist until the user logs out or closes the browser.
- **two-way communication** means that every time the browser/client makes a request, the server receives the request and cookies containing specific parameters and a unique Session ID which the server generates to identify the user.

Django Sessions

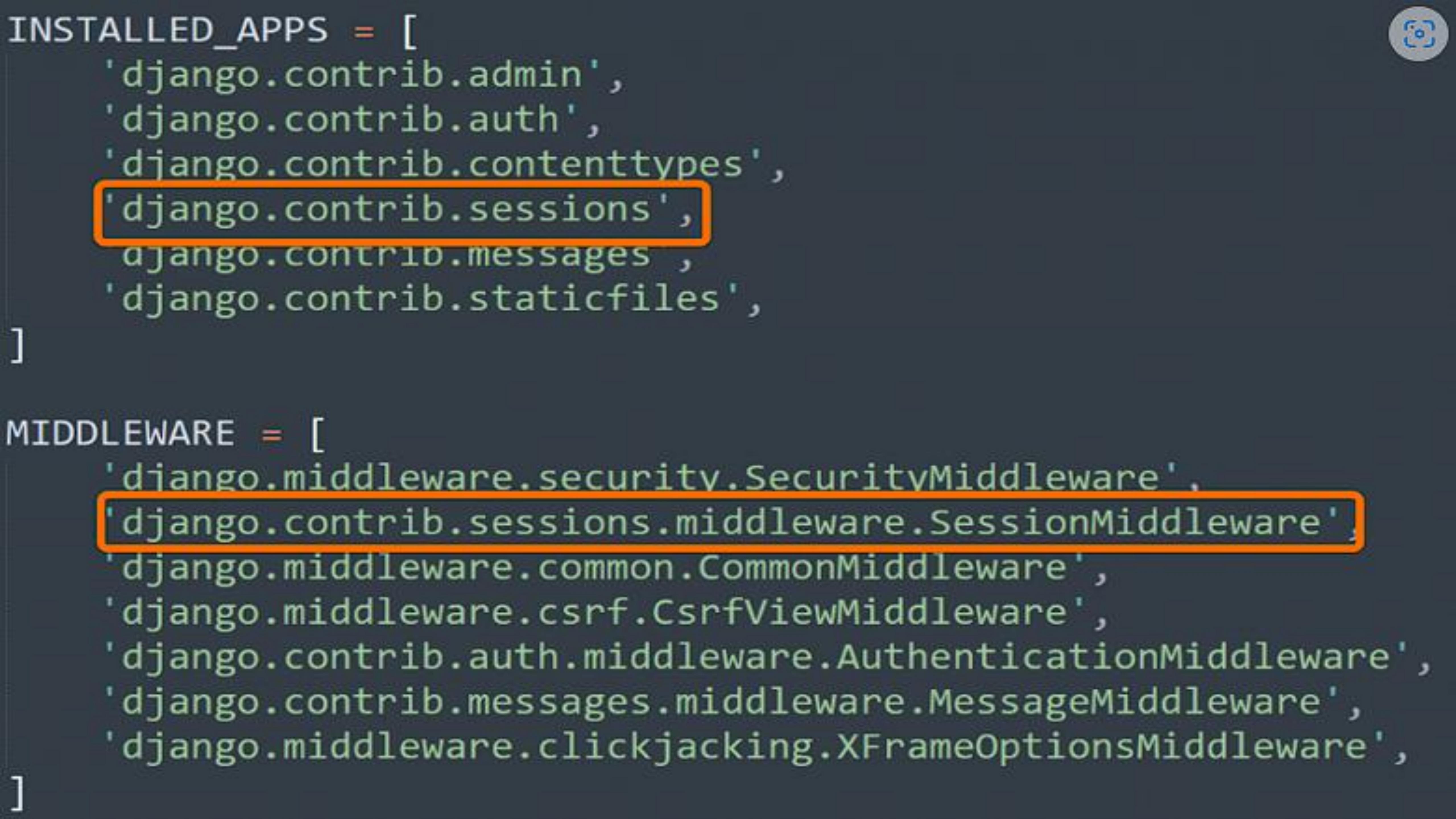
The Session ID **doesn’t change** for a particular session, but the website generates it every time a new session starts.

Important Session Cookies containing these Session IDs deletes when the session ends. But, this won’t have any effect on the cookies which have fix expire time.

Django Sessions

Django considers the importance of sessions over the website and therefore provides you with **middleware** and **inbuilt app** which will help you generate these session IDs without much hassle.

The middleware.SessionMiddleware is responsible for generating your unique Session IDs. You will also require django.contrib.sessions application, if you want to store your sessions on the database.



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

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

Django Sessions

The django.contrib.sessions application is present in the list of INSTALLED_APPS in settings.py file.

For working on sessions, you will need to check whether your browser supports cookies or not.

Django Sessions - How to Create, Use and Delete Sessions - DataFlair (data-flair.training)

Django Sessions

Configuring Session Engine

There are many ways by which you can configure sessions engine. It is an important decision where storage options can provide you better speed and response time.

Django lets you choose between these options but doesn't limit your choices.

Django Sessions

Configuring Session Engine

1. By Database-backed Sessions

We have used this method until now, as we have ***django.contrib.sessions*** application inside our INSTALLED_APPS list in settings.py file.

This is standard implementation and covers many security loopholes. It also provides you with lots of features to use them directly with view functions and models in Django framework.

Django Sessions

Configuring Session Engine

2. By Cached Sessions

If you want to improve the performance of the website, you can use cache-based sessions. There are several practices which you can perform here, like setting up multiple caches. Django provides you with more session engines:

django.contrib.sessions.backends.cache

Above engine stores data directly into the cache.

django.contrib.sessions.backends.cache_db

This engine, on the other hand, is used when you want persistent sessions data with your server using write-through cache. It means that everything written to the cache will also be written to the database.

Django Sessions

Configuring Session Engine

3. By File-based Sessions

When you want to store your sessions data in a file-based system, then you can use this engine that Django provides.

django.contrib.sessions.backends.file

You will also need to check whether your web server has permission to read/ write the directory in which you are keeping your sessions file.

Django Sessions

Configuring Session Engine

4. By Cookie-based Sessions

We don’t recommend this one as sessions were made for not storing data in cookies in the first place, but still, developers use it according to their need.

django.contrib,sessions.backends.signed_cookies

This allows your session data to store with the help of Django’s cryptographic signing tool and the secret key. It poses all the problems of cookies, the major one is that the session_cookies are signed but not encrypted; therefore they can be read by the client. Also, it has performance issues as the cookie size does affect the request size.

Django Advanced Topics

- Django Sessions
- **Django Request and Response**
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Request and Response

All backend technologies and web APIs are based on one system that is The Request-Response Cycle. The system is responsible for data interchange between client and servers. Even the new technologies based on micro-services use the Request/Response cycle.

Request/Response objects are transmitted over the web. Content like images, HTML, CSS, JavaScript is all Response Objects.

Django Request and Response

How does Web Work?

- any application over the internet works by passing information.
- This information is often passed as objects.
- The objects mentioned here comprises of various files or properties.
- These objects can contain files as attachments.

It is a very similar concept to emails.

Django Request and Response

These interchanges between various servers and client machines following a set of rules.

That set of rules is called **HTTP**.

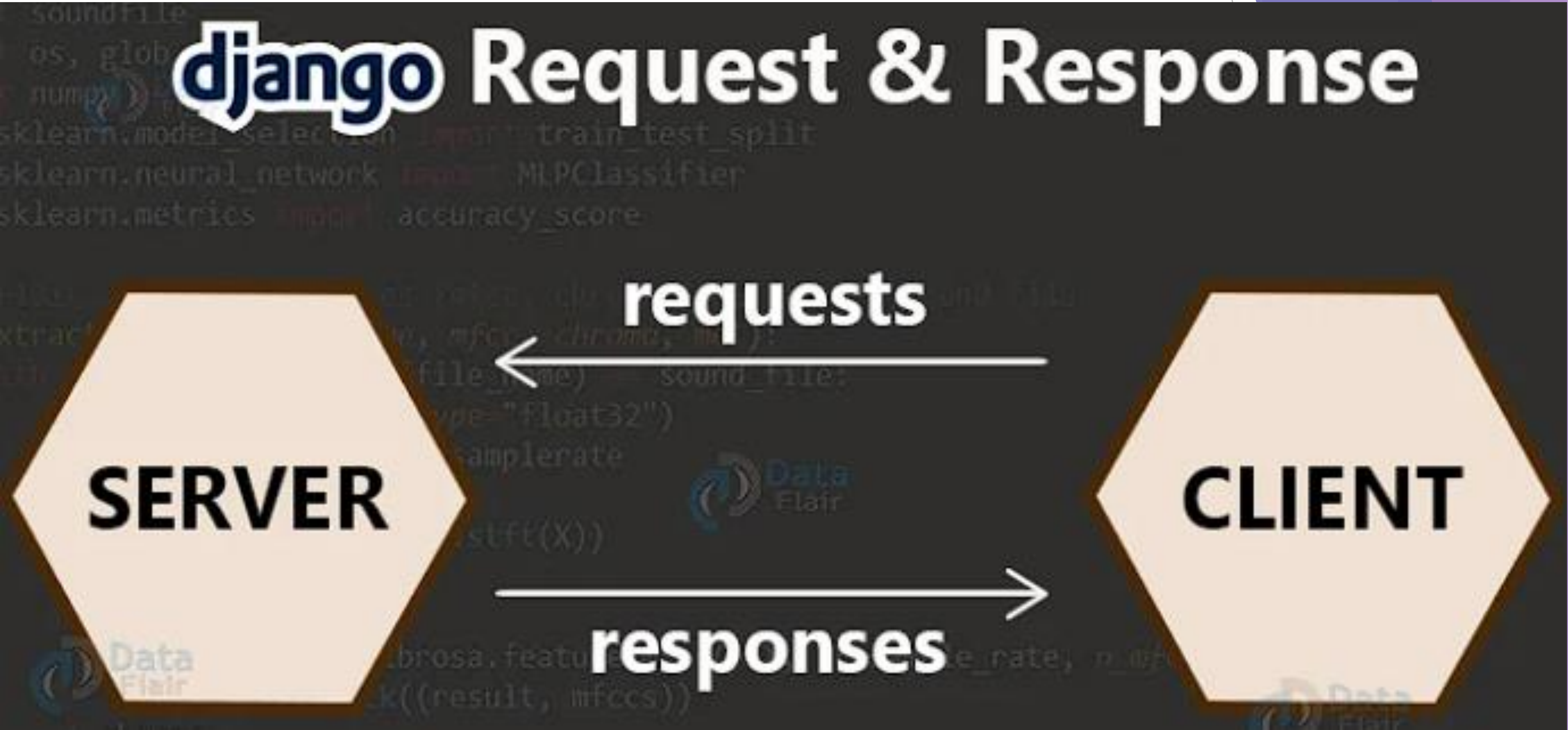
HTTP stands for **HyperText Transfer Protocol**. It's simply a set of rules, a common standard followed by devices to connect to each other over the internet.

Django Request and Response

The information sent or received is given the name of **request and response**.

Request Objects contain the request made by the client. The server responds with the appropriate information according to Request Object.

Django Request and Response



Django Request and Response

Request

- a “request made by the client” to the server.
- smaller in size than response.
- Any URL that gets searched is a request.
- also give information to the server.

The forms we fill, the images we upload to the server are also Requests. When we upload this information, it is transferred as attributes of the Request object. The server can then access that information and call its own functions.

Django Request and Response

Response

- ➢ generally heavier in bandwidth than Request.
- ➢ usually contain HTML, CSS, JavaScript, image, video files, etc.

Django Request and Response

Any response or request objects have a part to them called **HTTP headers**.

- ➢ contain various information regarding the response.
- ➢ consider it as metadata of response objects.
- ➢ tells the browser that from where the response came and what type of data it contains.

Django Request and Response

Django Request-Response Cycle

Django can be considered as an application on the server. Its main task is to process the request received by the server. Then it calls functions and provides a response. This is a brief of how Django interprets Request and provides Response.

Whenever a request comes into Django, it is handled by middlewares. When the Django server starts, the first thing it loads after settings.py is middlewares.

Django Request and Response

Django Request-Response Cycle

- The Request is processed by various **middlewares** one at a time.
- will be passed through the **security** middleware.
- the **authentication** requests are handled by authentication middleware.
- passed to the **URL Router**. The URL router simply extracts the URL from the request and will try to match it to defined urls.

Django Request and Response

Django Request-Response Cycle

- Match URL - > corresponding **view function** is called. The view function will get various attributes and other URL parameters. Also, the view function will be able to access files from the requests. These Requests are considered to be HttpRequest class objects.
- The requests module is a **Python module** that provides various methods for Request objects.

Django Request and Response

Django Request-Response Cycle

- Once, the view function has been executed, it’s time to give a **response**. The response is given in the form of HttpResponse. The response is not limited to that. The Response can be PDF, JSON, CSV. That is one of the features of Django. It provides built-in support to provide responses of various types.

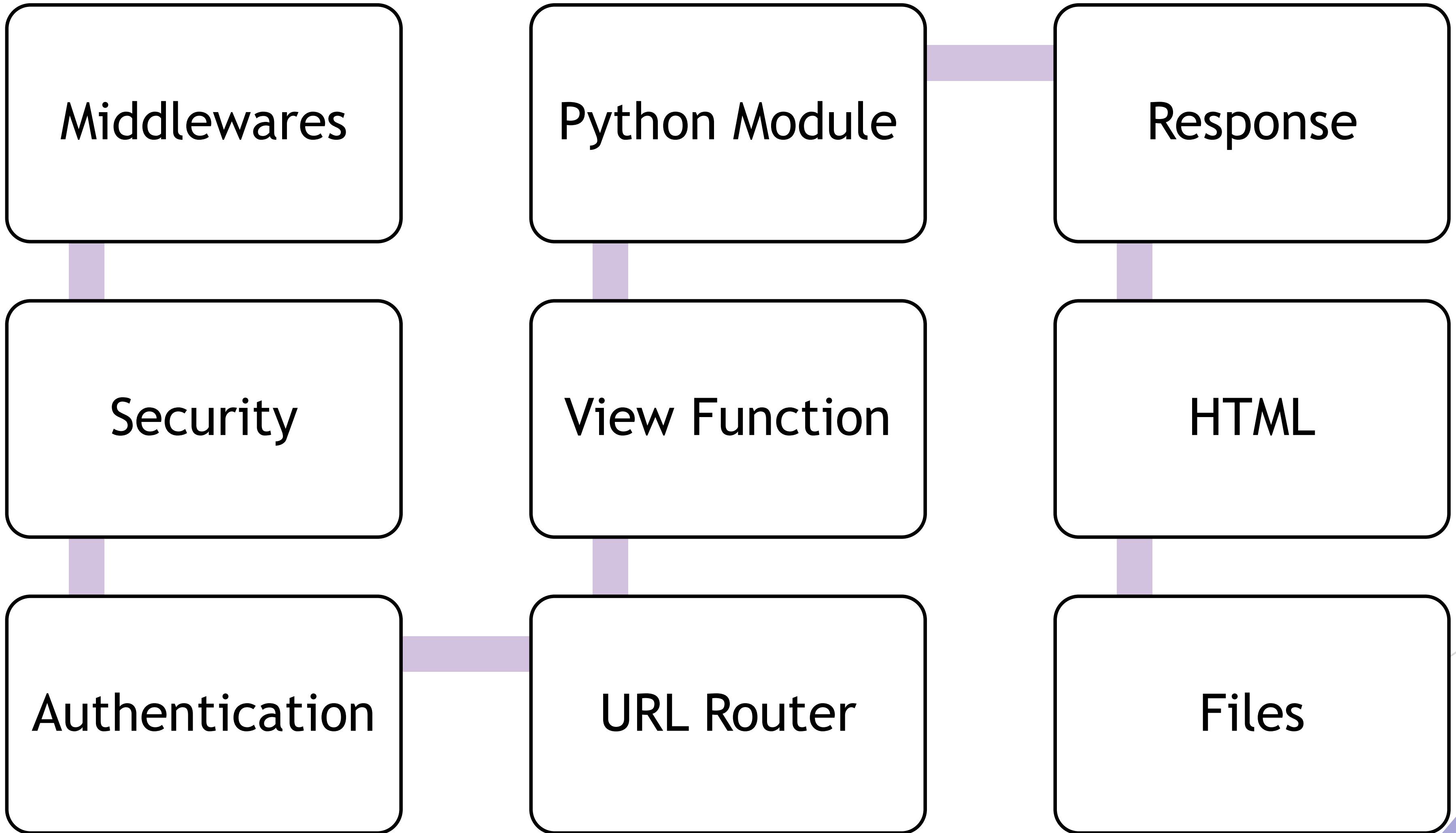
Django Request and Response

Django Request-Response Cycle

- When the response is a render, it will look for the **HTML**. The HTML page to be the server is processed by Django Templating Engine.
- Once that completes, the Django simply sends the **files** as any server would. That response will contain the HTML and other static files.

Django Request and Response

Django Request-Response Cycle



Django Advanced Topics

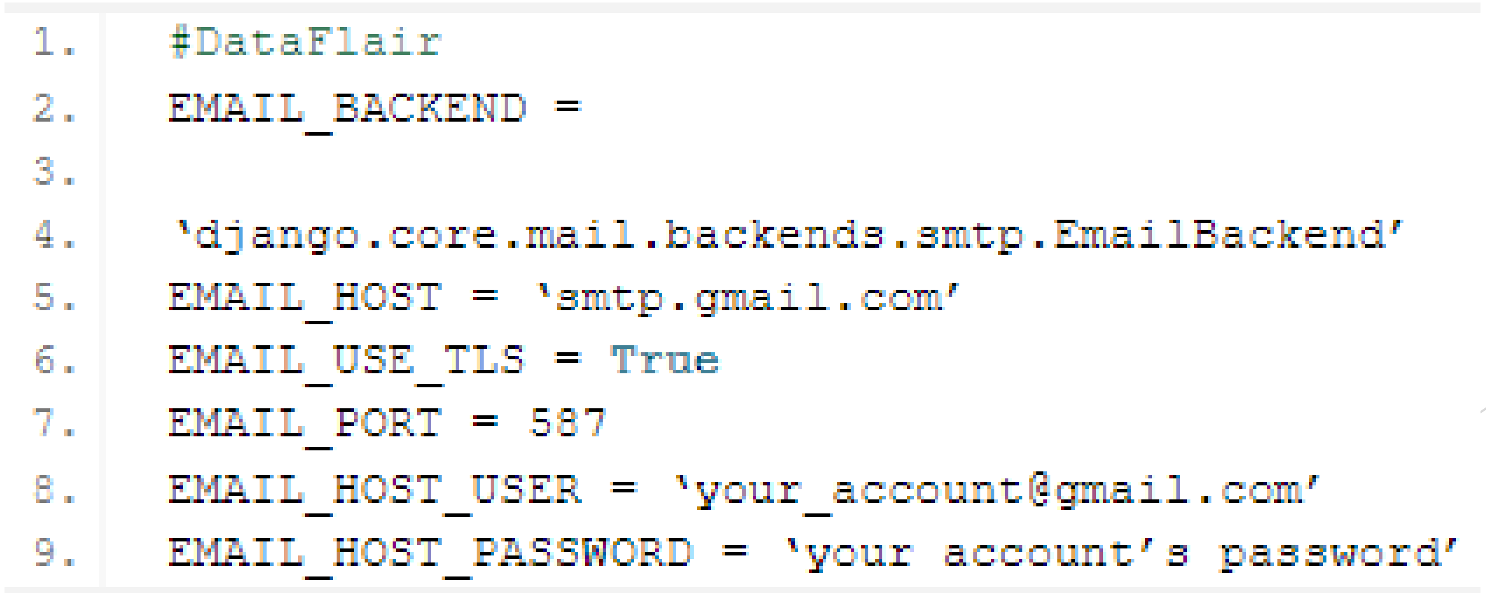
- Django Sessions
- Django Request and Response
- **Django Emails**
- Django Migrations
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Emails

Sending emails via websites is a great feature to have as it enhances the User Experience.

Settings for Sending Emails in Django

The value changes according to the type of email services. These settings are to be added in the **settings.py** file of the project.

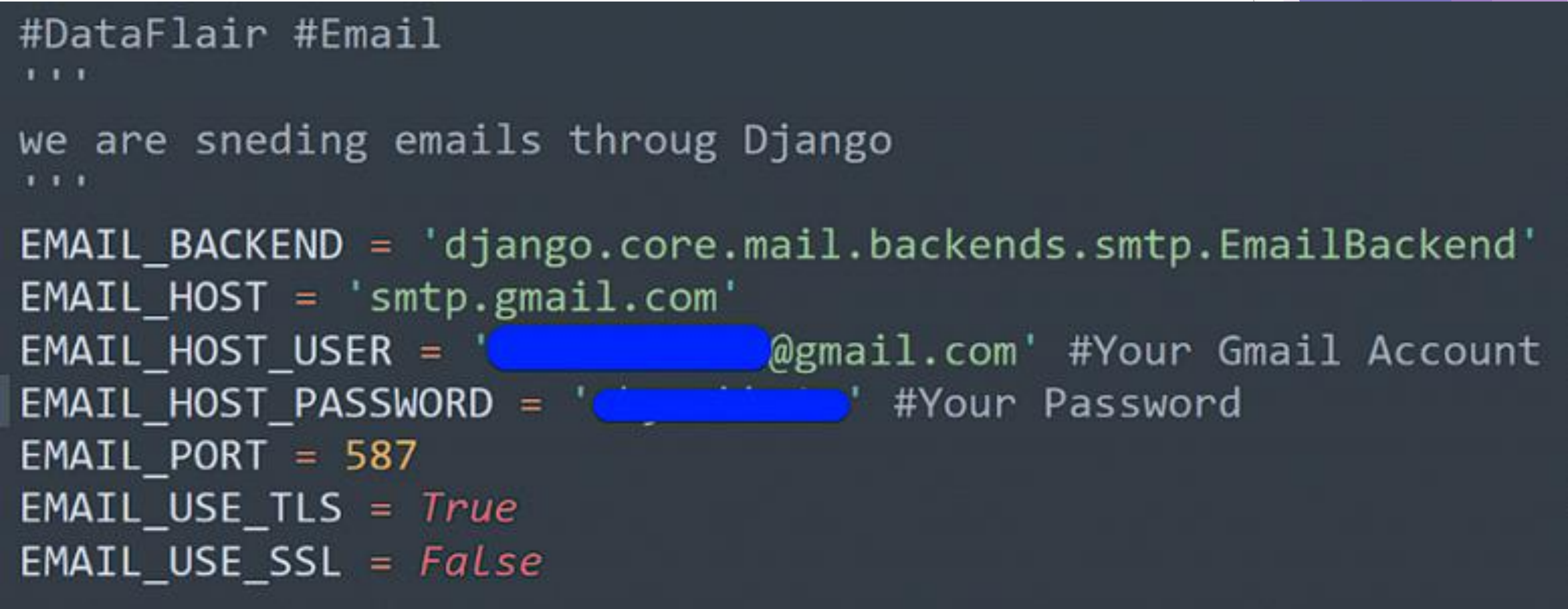


```
#DataFlair
EMAIL_BACKEND =

'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_USE_TLS = True
EMAIL_PORT = 587
EMAIL_HOST_USER = 'your_account@gmail.com'
EMAIL_HOST_PASSWORD = 'your account's password'
```

Django Emails

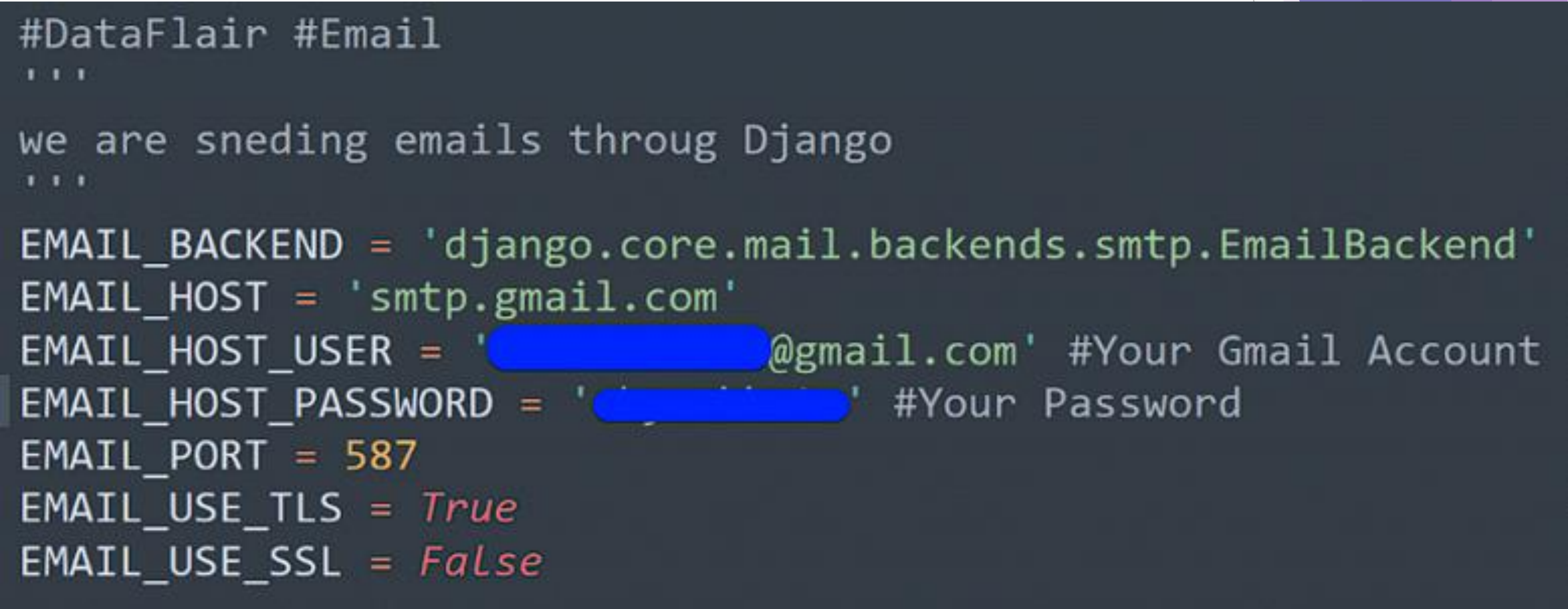
Settings for Sending Emails in Django



```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

EMAIL_BACKEND

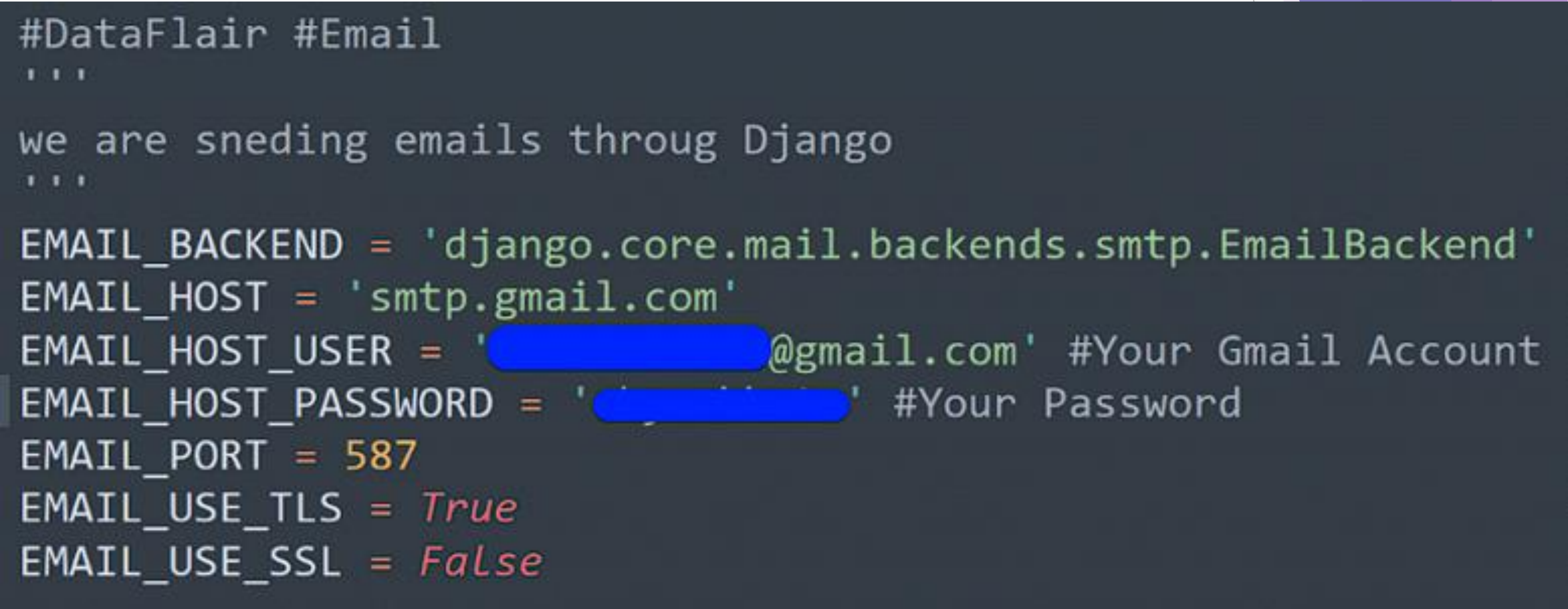
This setting specifies the backend that we are going to use for sending an email in Django. There are multiple backends that Django provides.



```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

EMAIL_HOST

This setting is to specify your email service provider. We are using the setting for Gmail.



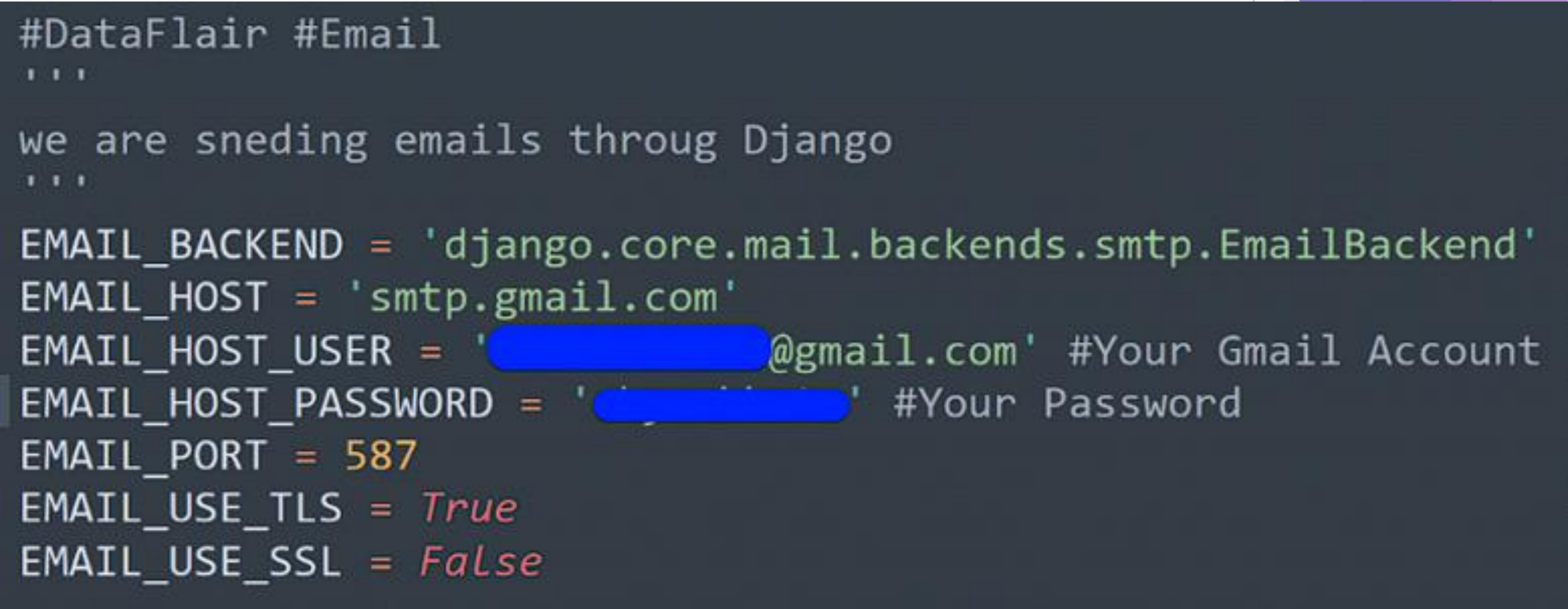
```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

EMAIL_HOST_USER

It is the account name from which we will send an email.

EMAIL_HOST_PASSWORD

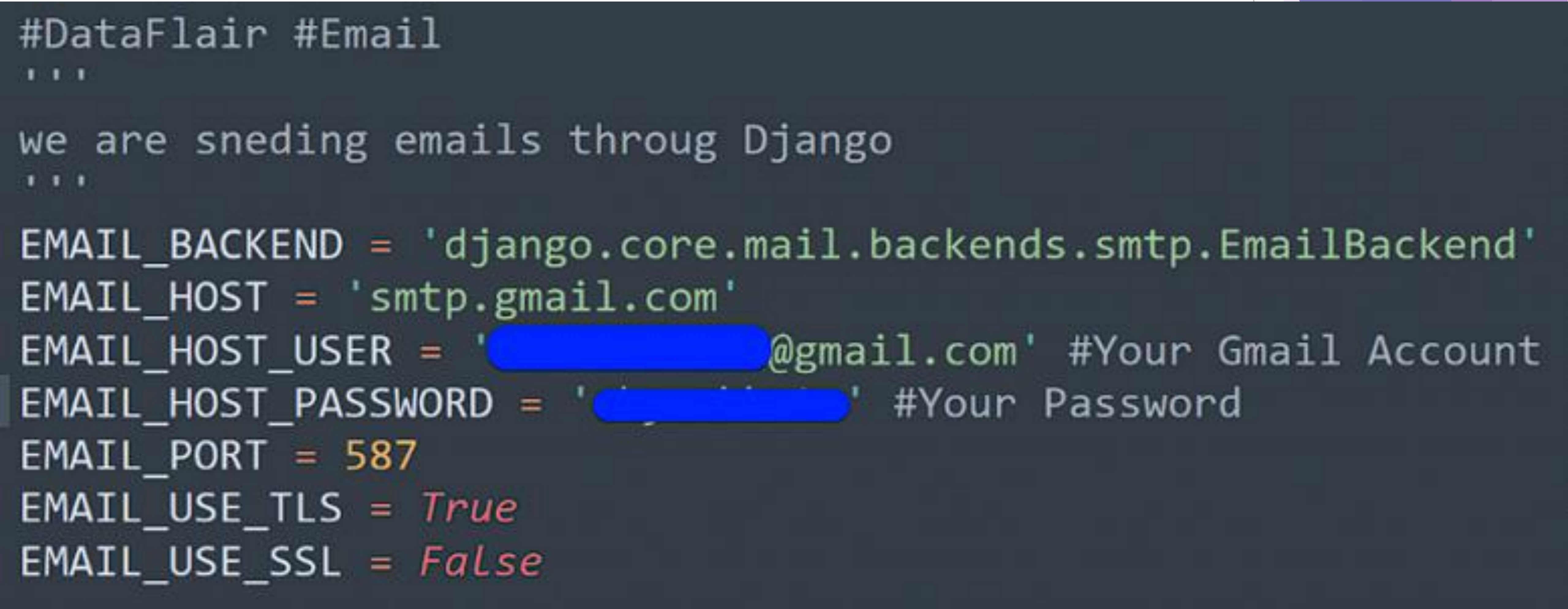
The password of the email account that we are using.



```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

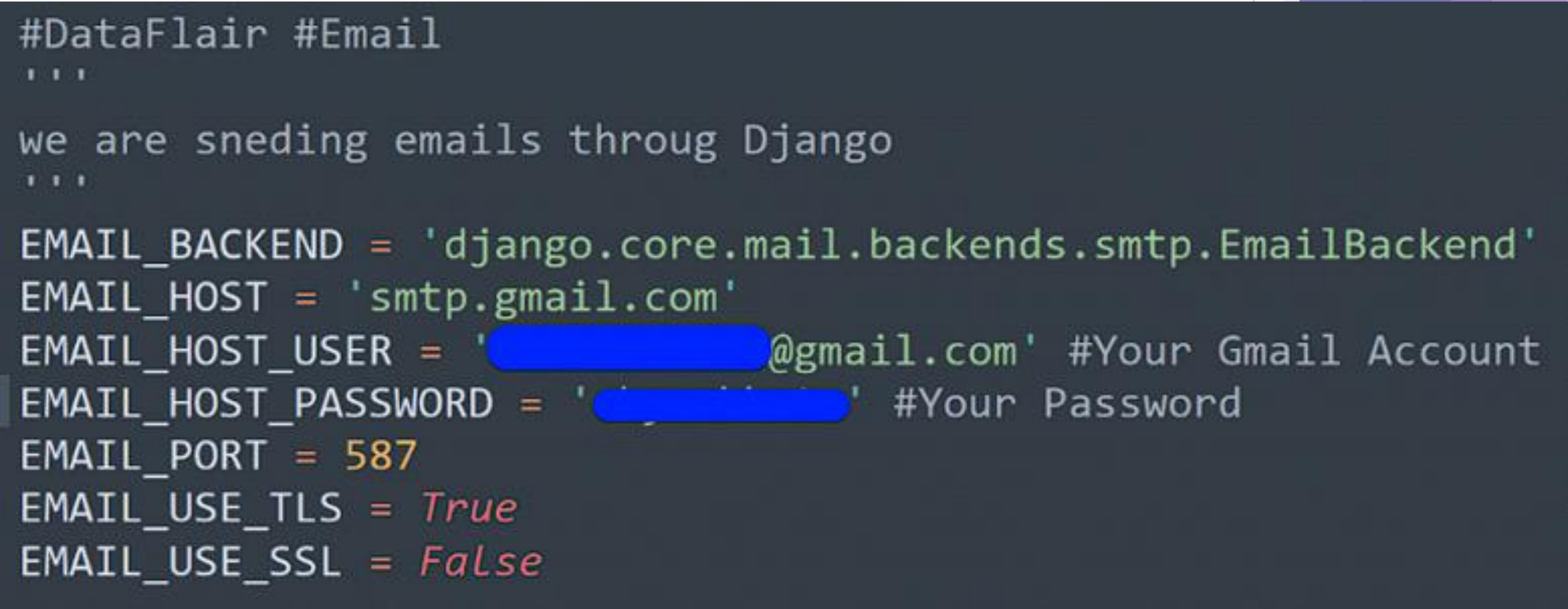
EMAIL_PORT

This is the setting for Gmail, you can get yours on the web. It is the port used by the SMTP (Simple Mail Transfer Protocol) server.



```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

EMAIL_USE_TLS
This setting specifies whether the Email uses a TLS (Transport Layer Security) connection or not. It is True for Gmail.

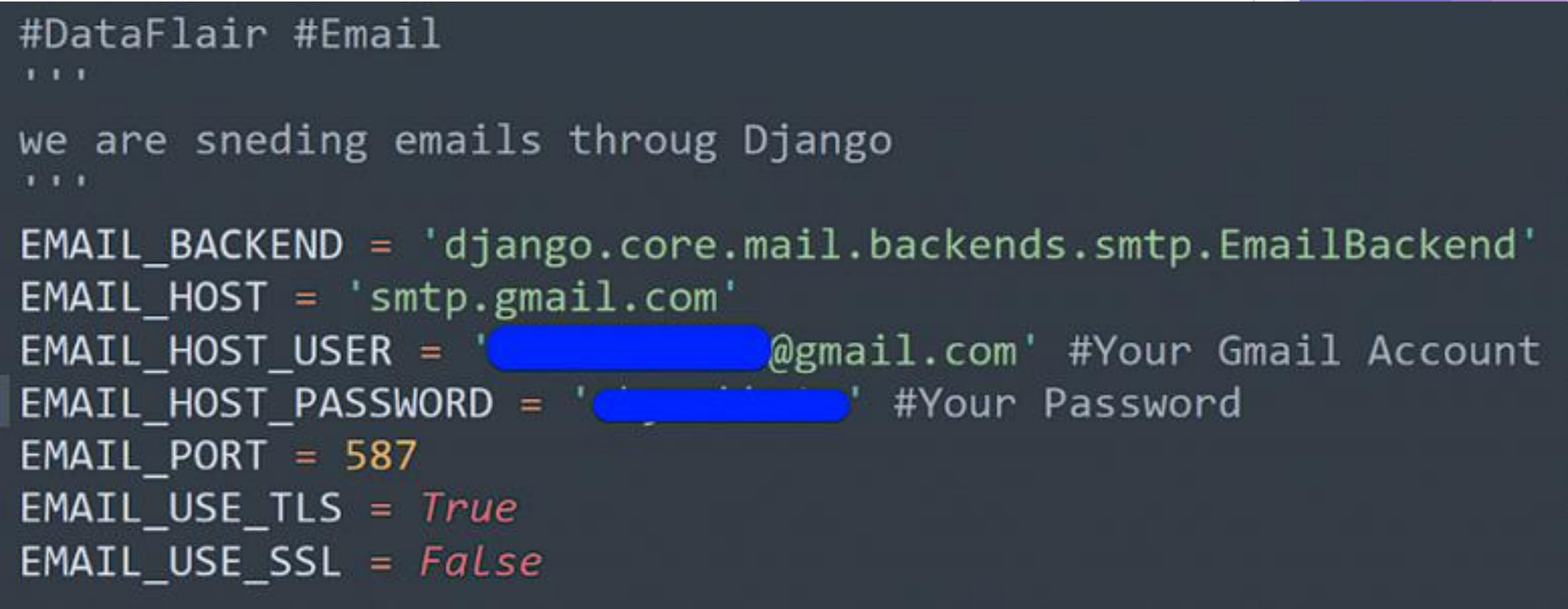


```
#DataFlair #Email
'''
we are sneding emails throug Django
'''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

EMAIL_USE_SSL (Secure Sockets Layer)

False for Gmail.

[How to Send Emails in Django - Elite Guidelines for Beginners - DataFlair (data-flair.training)]



```
#DataFlair #Email
'''

we are sneding emails throug Django
'''

EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_HOST_USER = '[redacted]@gmail.com' #Your Gmail Account
EMAIL_HOST_PASSWORD = '[redacted]' #Your Password
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_USE_SSL = False
```

Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- **Django Migrations**
- Django ORM
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Migrations

Django is an awesome framework. It abstracts the most trivial tasks and allows for rapid development. Since Django has an active community, they are constantly adding new features to it. These features give Django an edge in web development.

A database is a collection of data arranged in an efficient way. It ensures fast and secure storage, access and update of data.

A relational database is a collection of tables having a number of columns and rows.

Django Migrations

SQL (Structured Query Language) is a very popular language for databases. Almost all the popular databases use SQL. The databases like PostgreSQL, MySQL, SQLite, etc. are some of many examples.

The majority of SQL databases are relational and are extensively used in the industry.

Django Migrations

Django migrations are nothing but Python Files. They contain Python code which is used by Django ORM to create and alter tables and fields in the database.

Django implements Models and changes to them on the database via migrations. Django generates migrations automatically. This is a key feature and you should also take care of these. Migrations are not meant to be corrected in normal conditions.

Django Migrations

Key Benefits of Django Migrations

- Creating tables without SQL
- Models and Database Schema are always synchronized
- Don’t Repeat Yourself (DRY) philosophy
- Easier Version Control for Database Scheme

Django Migrations and Database Connectivity - An Excellent concept made Easy! - DataFlair (data-flair.training)

Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- **Django ORM**
- Django Caching
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django ORM

ORM is an acronym for the object-relational mapper. The ORM’s main goal is to *transmit data* between a relational database and application model. The ORM automates this transmission, such that the developer need not write any SQL.

ORM, as from the name, maps objects attributes to respective table fields. It can also retrieve data in that manner.

Django ORM Tutorial - The concept to master Django framework - DataFlair (data-flair.training)

Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- **Django Caching**
- Django Admin Customization
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Caching

In today’s era of web development, every user wants their requests on the website to be responded ASAP. There are certain methods which will make your website faster. Faster the website, greater the users it will attract.

Let’s take an example of **WhatsApp**. This popular messaging application works on very low speed in bytes but you can receive text messages instantly because it uses different technology due to which we get the importance of faster response.

Django Caching

Caching is one of those methods which a website implements to become faster. It is cost efficient and saves CPU processing time.

Dynamic websites are a collection of webpages which are generated dynamically.

Dynamic websites generate highly personalized webpages. They come with tons of features. Some examples of websites are Spotify, Google, Facebook, Instagram, etc.

A very good example can be Google Feed. The personal news feed on google app is generated dynamically. Its contents change in every 2-3 hrs. The updating process takes time if you refresh.

Django Caching

Most of the websites on the Internet are becoming more and more dynamic and that makes it necessary for web developers to create faster websites. **Caching** comes in there. It provides a cost-effective and working solution. The more processing time a webpage takes, the more caching can improve its performance.

There are other solutions as well:

- Installing better server hardware
- Installing the better server software
- Improving database
- Writing efficient code, etc.

Django Caching

Caching is the process of storing recently generated results in memory. Those results can then be used in future when requested again.

The computer memory works in a similar manner. CPU stores some files in the cache Memory. And, when CPU needs them again it looks for those files in Cache Memory first. Since Cache Memory is fast, the processing time improves.

Django Caching

Caching on Server

Suppose we have a dynamic website. When a user request for some page:

1. A Dynamic Page generates
2. A webpage serves and gets stored in Cache Space

Cache Space is a location where you temporarily store data to be retrieved for later use.

3. Multiple users request for the same page
4. Server responses with Cached Webpage
5. Deletion of data in Cache Space

Django Caching

Some important points:

- **CPU time of Webpage -**caching subtracts processing time from response time. So, if our website serves static pages, there may not be any improvement. Therefore, caching is mostly used for dynamic webpages.
- **Memory type used for Cache Space**

Caching space is also a factor to improve speed. There can be **three types of Cache Spaces**:

- ➢ Main Memory
- ➢ Database Tables
- ➢ File-System or local directories

Django Caching

Some important points:

- **Expiry time of cached data**

Cached data should exist for a certain period of time. If the data gets deleted too early, there will be no utilization of caching. Also, if the data doesn’t get deleted, then it will waste cache memory. It depends on the type of resource and its demand by the users.

[Django Caching - It's Easy if you do it in the Smart Way! - DataFlair (data-flair.training)]

Django Advanced Topics

- Django Sessions
- Django Request and Response
- Django Emails
- Django Migrations
- Django ORM
- Django Caching
- **Django Admin Customization**
- Django Exceptions & Error Handling
- AJAX in Django
- Django Web Hosting
- Django CMS
- Django REST Framework
- Django Logging

Django Admin Customization

Django Admin is one of the most important tools of Django. It’s a full-fledged application with all the utilities a developer need. Django Admin’s task is to provide an interface to the admin of the web project.

Django Admin’s aim is to provide a user interface for performing CRUD operations on user models and other functionalities like authentication, user access levels, etc. These all are functions that make admin more productive.

Django Admin Customization

1. Register/ Unregister models from admin
2. Changing the title of Django Admin
3. ModelAdmin Class (a representation of user-defined models in the admin panel)
4. Customizing List Display in Django Admin
5. Adding a filter in admin

Django Admin Customization - Step-by-step tutorial for novice learners - DataFlair (data-flair.training)

Applications???

Any questions???

REFERENCES:

- Django Tutorial. (2022). https://data-flair.training/blogs/django-tutorial/
- Django Tutorial. (2020). https://www.geeksforgeeks.org/django-tutorial/
- Django Tutorial. (2022). https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
- Django Tutorial. (n.d.). https://www.tutorialspoint.com/django/index.htm
- Django Tutorial. (n.d.). https://www.w3schools.com/django/index.php
- Getting started with Django. (n.d.). https://www.djangoproject.com/start/