**settings.py**

**from pathlib import Path** : Python pathlib module provides an object-oriented way to handle filesystem paths. Here’s how we can use it to perform common filesystem operations more efficiently than with the older os and os.path modules. **[Importing the Main Class]**

**BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent --** is a common way to set a variable called BASE\_DIR to the absolute path of the root directory of your project, by using the pathlib module to navigate up the directory structure from the current file's location to its grandparent directory.

**SECRET\_KEY = 'django-insecure-0d!!1ktg0ao-asfecf8!e7p46+bw227^wym$4-2x5nj2j2qry4' :** Django, a SECRET\_KEY is used for cryptographic signing, that is, to generate hashes and tokens. If any other person has access to our SECRET\_KEY , then they can generate our hashes and tokens. Thus, it's very important for us to protect the SECRET\_KEY .

* Encrypt and decrypt: Ensure confidentiality and prevent unauthorized access to protected information
* Generate hashes and tokens: Used for cryptographic signing
* Sign data: Detect tampering with data sent over the wire

**DEBUG=True** is a way to tell Django that we are currently in the development phase and the runserver can serve the static files. When we set DEBUG=False, the Django default server won't work, it requires the ALLOWED\_HOSTS and some production configuration.

**Topic is ALLOWED\_HOSTS = []** is a setting within your project's settings.py file that defines a list of domain names (or hostnames) that your Django application is allowed to serve, acting as a security measure to prevent malicious "HTTP Host header attacks" by only accepting requests from specified domains; essentially, if a request comes from a hostname not listed in ALLOWED\_HOSTS, Django will raise an error, preventing unauthorized access.

* **Security feature:**

It's crucial to set ALLOWED\_HOSTS to protect your application from potential attacks where a malicious user could try to manipulate the "Host" header in a request to target unintended domains or applications on your server.

* **Empty list warning:**

Leaving ALLOWED\_HOSTS as an empty list is generally not recommended as it effectively allows any hostname to access your application, which can be a major security risk.

**ALLOWED\_HOSTS = ['www.mydomain.com', 'mydomain.com', 'api.mydomain.com']**

**INSTALLED\_APPS = [**

**'django.contrib.admin',**

**'django.contrib.auth',**

**'django.contrib.contenttypes',**

**'django.contrib.sessions',**

**'django.contrib.messages',**

**'django.contrib.staticfiles',**

**]**

**MIDDLEWARE = [**

**It should go near the top of the list if you’re going to turn on the SSL redirect as that avoids running through a bunch of other unnecessary middleware.**

**'django.middleware.security.SecurityMiddleware',**

**Before any middleware that may raise an exception to trigger an error view (such as [PermissionDenied](https://docs.djangoproject.com/en/5.1/ref/exceptions/" \l "django.core.exceptions.PermissionDenied" \o "django.core.exceptions.PermissionDenied)) if you’re using**[**CSRF\_USE\_SESSIONS**](https://docs.djangoproject.com/en/5.1/ref/settings/#std-setting-CSRF_USE_SESSIONS)**.**

**'django.contrib.sessions.middleware.SessionMiddleware',**

Before any middleware that may change the response (it sets the Content-Length header). A middleware that appears before CommonMiddleware and changes the response must reset Content-Length.

Close to the top: it redirects when APPEND\_SLASH or PREPEND\_WWW are set to True.

After SessionMiddleware if you’re using CSRF\_USE\_SESSIONS.

**'django.middleware.common.CommonMiddleware',**

**'django.middleware.csrf.CsrfViewMiddleware',**

**Before any view middleware that assumes that CSRF attacks have been dealt with.Before RemoteUserMiddleware, or any other authentication middleware that may perform a login, and hence rotate the CSRF token, before calling down the middleware chain.After SessionMiddleware if you’re using CSRF\_USE\_SESSIONS.**

**After SessionMiddleware: uses session storage.**

**'django.contrib.auth.middleware.AuthenticationMiddleware',**

**After SessionMiddleware: can use session-based storage.**

**'django.contrib.messages.middleware.MessageMiddleware',**

**'django.middleware.clickjacking.XFrameOptionsMiddleware'**

**]**

**ROOT\_URLCONF = 'home.urls'**

In Django, "ROOT\_URLCONF = 'home.urls'" means that the primary file responsible for mapping URLs to views in your application is located at a Python module named "urls" within the "home" app directory, and Django will use this file to determine how to handle incoming URL requests based on the patterns defined within it; essentially, it sets the root URL configuration for your project to the "urls.py" file inside the "home" app.

**"ROOT\_URLCONF": Explain-----**

This is a Django setting variable that specifies the path to the file containing your URL patterns.

**home.urls:**

This indicates that the file named "urls.py" within the "home" app directory is the file that contains your URL patterns.

How it works:

**When a user visits a URL:**

Django receives the URL request.

**Consult the ROOT\_URLCONF:**

Django looks at the "ROOT\_URLCONF" setting to find the correct URL configuration file (in this case, "home.urls").

**Match URL patterns:**

Django then checks the patterns defined within the "urls.py" file in the "home" app to see which pattern matches the requested URL.

**Execute view function:**

If a match is found, Django calls the corresponding view function associated with that URL pattern to generate the response.

Important considerations:

**File structure:**

Ensure that you have a file named "urls.py" inside your "home" app directory.

**URL patterns:**

Within the "urls.py" file, you will define URL patterns using Django's path function, connecting each pattern to a corresponding view function.