Project Django - Blog

- Create A New GitHub Repository

> GitHub Repository

- Setup Venv to Create Project

> *python -m venv venv*
>
> *venv\Scripts\activate.bat*

- Create Project

> - pip install django
> - django-admin -> django-admin --version
> - django-admin startproject `Django_Blog_Project`
> - cd `Django_Blog_Project` -> python manage.py runserver

> - **Create `.gitignore`**
>
> -> venv/ -> db.sqlite3

> - **Create `requirements.txt` in `Django_Blog_Project`**
>
> -> pip freeze > requirements.txt

- Connect GitHub

```
-> git init
-> git add .
-> git commit -m "first commit"
-> git branch -M main
-> git remote add origin
git@github.com:ericarthuang/Copy_Learning_Django.git
-> git push -u origin main
```

- Creating apps and Register App & URL

> - Creating app
>
> -> py manage.py startapp blog_app
>
> - Register App

> -> In `djangotoec2_main/settings.py`, add `blog_app` in the `INSTALLED_APPS list`
> -> `Open the file blog_app/views.py`
>
> - Register URL
>
> -> Create a URLconf file called `urls.py` in `blog_app folder`
> -> Point the root URLconf at the `blog_app.urls` module
> -> In `djangotoec2_main/urls.py`, add `import for django.urls.include` and insert an include() in the urlpatterns list

- urls, views, templates, and static

> - In Django_Blog_Project/urls.py, insert an path in the urlpatterns list
>
> -> path('', include('blog_app.urls')),
>
> - In Django_Blog_Project/views.py, define the index(requrest) for index.html
>
> -> Can define multiple html in Django_Blog_Project/views.py to link htmlfiles in templates

> ### templates
>
> - Using `templates` folder to keep htmlfiles
>
> -> Create `templates` folder in 'blog_app' folder
> -> Create htmlfiles in the 'templates' folder
> -> *Can create multiple folders for multiple apps -> `blog_app` folder -> `templates` folder -> `blog_app` folder -> htmlfiles*
> -> *NOTICE: The Name of Folder should be sames as app**
>
> - Create `base.html` for htmlfiles

> ### static
>
> - Using `static` folder to keep css, images, and other static files
>
> -> Create `static` folder in 'blog_app' folder
> -> Create `static` files in `static` folder
> -> *Can create multiple folders for multiple apps*
> -> *`blog_app` folder -> `static` folder -> `blog_app` folder -> css, images, and other static files*
> -> *NOTICE: The Name of Folder should be sames as app**
>
> - Put {% load static %} in `base.html`
>
> -> `<link rel="stylesheet" href="{% static 'blog_app/main.css' %}">`

- Admin Page

> - CMD: python manage.py makemigrations
> - CMD: python manage.py migrate
> - CMD: python manage.py createsuperuser
>
> -> go to `http://127.0.0.1:8000/admin/` for logining
> -> go to `http://127.0.0.1:8000/admin/auth/user/1/change/` to know the hassing password

- Database and Migrations - Sqlite3

> - **`Using DB Browser(SQLite) to view the database`**
> - blog_app/models.py

```
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User
class Post(models.Model):
   title = models.CharField(max_length=100)
   content = models.TextField()
   date_posted = models.DateTimeField(default=timezone.now) #
don't use timezone.now()
   #Foreign Key
   author = models.ForeignKey(User, on_delete=models.CASCADE)
   def __str__(self):
       return self.title
```

> - CMD: python manage.py makemigrations
> - CMD: python manage.py migrate

> - go to app.views.py

```
from .models import Post
context = {
        'posts': Post.objects.all()
   }
```

> ### setup models in the `Site administration`
>
> - blog_app/admin.py

```
from .models import Post

-> *admin.site.register(Post)*
```

- User Registration

> - Creating app
>
> -> CMD: Python manage.py startapp user_app

> - Register App
>
> -> In Django_Blog_Project/settings.py, add 'user_app' in the 'INSTALLED_APPS' list

> - Register URL
>
> -> In Django_Blog_Project/urls.py
>
> ```
> from user_app import views as user_views
> -> insert an path in the urlpatterns list
> -> path('register/', user_views.register, name='register'),
> ```

> - Create Views
>
> -> Open the file user_app/views.py
>
> -> Using `UserCreattionForm` to setup user register form
>
> - Create `templates` folder in `user_app` folder
>
> -> Create `user_app` folder in `templates` folder
>
> -> Create `register.html` in `templates/user_app` folder
>
> -> create `csrf_token` in `register.html`
>
> ```
> <form method="POST">
>     {% csrf_token %}
>     {{ form }}
> </form>
> ```

> ## Enhance the register process
>
> - create `forms.py` in `user_app` folder
>
> ```
> from django import forms
> from django.contrib.auth.models import User
> from django.contrib.auth.forms import UserCreationForm
>
> class UserRegisterForm(UserCreationForm):
>    email = forms.EmailField()
>    class Meta:
>        model = User
> fields = ['username', 'email', 'password1', 'password2']
> ```

> ## Message
>
> - user_app/views.py
>
> -> `from django.contrib import messages`
>
> ```
> from django.shortcuts import render, redirect
> from django.contrib.auth.forms import UserCreationForm
> from django.contrib import messages
> from .forms import UserRegisterForm
>
> def register(request):
>     if request.method == "POST":
> ```

```
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for
{username}!')
            return redirect('blog-home')
    else:
        form = UserRegisterForm()
    return render(request, 'user_app/register.html', {'form':
form})
```

- link messages.tags with `base.html`

```
{% if messages %}
   {% for message in messages %}
       <div class="alert alert-{{ message.tags }}">
           {{ message }}
       </div>
   {% endfor %}
{% endif %}
```

using `crispy` to style the form

-> CMD: pip install django-crispy-forms

-> In Django_Blog_Project/settings.py, add 'crispy_forms' in the 'INSTALLED_APPS' list

-> In Django_Blog_Project/settings.py, add `CRISPY_TEMPLATE_PACK = "bootstrap4"`

-> In `register.html` -> {% load crispy_forms_tags %} -> {{ form|crispy }}

Login and Logout System

- In Django_Blog_Project/urls.py, insert an path in the urlpatterns list

```
from django.contrib.auth import views as auth_views
path('login/',
auth_views.LoginView.as_view(template_name="user_app/login.html"),
name='login'),
path('logout/',
auth_views.LogoutView.as_view(template_name="user_app/logout.html"),
name='logout'),
```

- In `user_app/templates`, create `login.html` and `logout.html`
- In Django_Blog_Project/settings.py, add `LOGIN_REDIRECT_URL = 'blog-home'`

- Profile

- In `Django_Blog_Project/urls.py`, insert an path in the urlpatterns list

-> `path('profile/', user_views.profile, name='profile')`

- In `user_app/views.py`:

```
def profile(request):
   return render(requeat, 'user_app/profile.htm')
```

- Create profile.html in `user_app/templates/user_app` folder

- link `profile.html` with `base.html`

```
<a class="nav-item nav-link" href="{% url 'profile'
%}">Profile</a>
```

check login when view the porfile

- In `user_app/views.py`:

```
from django.contrib.auth.decorators import login_requried

@login_required
def profile(request):
   return render(requeat, 'user_app/profile.htm')
```

- In `Django_Blog_Project/settings.py`

-> LOGIN_URL = "login"

user models

- In `user_app/models.py`:

```
from django.db import models
from django.contrib.auth.models import User
class Profile(models.Model):
    user = models.OneToOneField(User,
on_delete=models.CASCADE)
    image = models.ImageField(default='default.jpg',
upload_to='profile_pics')
   def __str__(self):
       return f'{self.user.username} Profile'
   def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
       img = Image.open(self.image.path)
       if img.height > 300 or img.width > 300:
           output_size = (300, 300)
           img.thumbnail(output_size)
           img.save(self.image.path)
```

- CMD: python manage.py makemigrations
- CMD: python manage.py migrate

setup models in the `Site administration`

- user_app/admin.py

```
from .models import Profile
```

> -> *admin.site.register(Profile)*
>
> - add profile with picture from `admin page`
>
> -> you will see the `profile_pics` folder will be creaged in the `Django_Blog_project` folder
> -> *We define the `profile_pics` folder in `user_app/models.py` Class Profile

> ### Change the folder to keep images
>
> - `pip install Pillow`
> - In `Django_Blog_Project/settings.py`
>
> -> MEDIA_ROOT = os.path.join(BASE_DIR, 'mdeia')
>
> -> MEDIA_URL = '/media/'
>
> -> delete profiles for retesting -> you will see the `media/profile_pics` folder in the `Django_Blog_project` folder

> ### Enhance `profile.html`

> - In `Django_Blog_Project/urls.py`
>
> ```
> from django.conf import settings
> from django.conf.urls.static import static
> urlpatterns = [
>    path('admin/', admin.site.urls),
>    ...]
> if settings.DEBUG:
>    urlpatterns += static(settings.MEDIA_URL,
>                               document_root=settings.MEDIA_ROOT)
> ```

- Combine User Register and Profile

> - create `signals.py` in the `user_app` folder
>
> from django.db.models.signals import post_save
>
> from django.contrib.auth.models import User
>
> from django.dispatch import receiver
>
> from .models import Profile
>
> ```
> @receiver(post_save, sender=User)
> def create_profile(sender, instance, created, **kwargs):
>     if created:
>         Profile.objects.create(user=instance)
> @receiver(post_save, sender=User)
> def save_profile(sender, instance, **kwargs):
>     instance.profile.save()
> ```

> - in `user_app/apps.py`
>
> ```
> from django.apps import AppConfig
> class UserAppConfig(AppConfig):
>     default_auto_field = 'django.db.models.BigAutoField'
> ```

```
    name = 'user_app'
    def ready(self):
        import user_app.signals
```

- Update User Profile

- go to `user_app/forms.py`

```
from .models import Profile
class UserUpdateForm(forms.ModelForm):
    email = forms.EmailField()
    class Meta:
        model = User
        fields = ['username', 'email']
class ProfileUpdateForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ['image']
```

- go to `user_app/views.py`

```
from django.contrib.auth.decorators import login_required
from .forms import UserRegisterForm, UserUpdateForm,
ProfileUpdateForm
@login_required
def profile(request):
    if request.method == 'POST':
        u_form = UserUpdateForm(request.POST,
instance=request.user)
        p_form = ProfileUpdateForm(request.POST,
request.FILES, instance=request.user.profile)
        if u_form.is_valid() and p_form.is_valid():
            u_form.save()
            p_form.save()
            messages.success(request, f'Your account have been
updated!')
            return redirect('profile')
    else:
        u_form = UserUpdateForm(instance=request.user)
        p_form =
ProfileUpdateForm(instance=request.user.profile)
    context = {
        'u_form': u_form,
        'p_form': p_form
    }
    return render(request, 'user_app/profile.html', context)
```

- put form section into 'profile.html'

```
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <fieldset class="form-group">
         <legend class="border-bottom mb-4">Profile
Info</legend>
         {{ u_form|crispy }}
         {{ p_form|crispy }}
    </fieldset>
    <div class="form-group">
         <button class="btn btn-outline-info"
type="submit">Update</button>
    </div>
</form>
```

> - control image size for uploading
>
> -> go to `user_app/models.py`
>
> ```
> from PIL import Image
>     def save(self):
>         super().save()
>         img = Image.open(self.image.path)
>         if img.height > 300 or img.width > 300:
>            output_size = (300, 300)
>            img.thumbnail(output_size)
>            img.save(self.image.path)
> ```

> - combine image to `home.html'
>
> -> go to blog_app/templates/home.html
>
> ```
> <img class="rounded-circle" src="{{
> post.author.profile.image.url}}">
> ```

- Reset Email and Password

> - Go to `Django_Blog_Project/urls.py`, insert an path in the urlpatterns list
>
> ```
> path('password-reset/',
>
> auth_views.PasswordResetView.as_view(template_name="user_app/password_r
>      name='password_reset'),
> ```

> - Create `user_app/templates/user_app/password_reset.html'

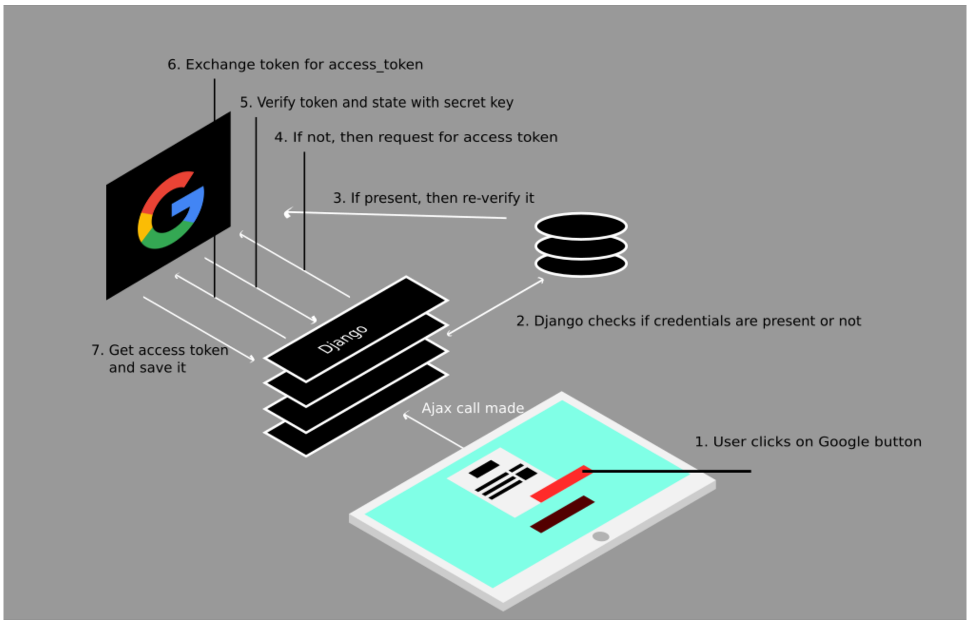
> - Go to `Django_Blog_Project/urls.py`, insert three paths in the urlpatterns list
>
> ```
> path('password-reset-confirm/<uidb64>/<token>/',
>
> auth_views.PasswordResetConfirmView.as_view(template_name="user_app/pas
>       name='password_reset_confirm'),
> path('password-reset/done/',
>
> auth_views.PasswordResetDoneView.as_view(template_name="user_app/passwc
>       name='password_reset_done'),
> path('password-reset-complete/',
> ```

```
auth_views.PasswordResetCompleteView.as_view(template_name="user_app/pa
      name='password_reset_complete'),
```

setup connection with Gamil

- go to `settings.py`

```
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_USE_TLS = True
EMAIL_HOST_USER = os.environ.get('********')
EMAIL_HOST_PASSWORD = os.environ.get('*******')
```


6. Exchange token for access_token
5. Verify token and state with secret key
4. If not, then request for access token
3. If present, then re-verify it
2. Django checks if credentials are present or not
7. Get access token and save it
Django
Ajax call made
1. User clicks on Google button

- Create, Update, and Delete Posts

Create Posts

- go to `blog_app/views.py`
- go to `blog_app/urls.py`
- Create `user_posts.html` in `blog_app/templates` folder
- Create `post_detail.html` in `blog_app/templates` folder

link `user_posts.html` and `post_detail.html` and `home.html`

- in `home.html`
- in `user_posts.html`

> - in `post_detail.html`
> - Create `post_form.html` in `blog_app/templates` folder for creating post
> - modify `blog_app/views.py` and `blog_app/urls.py` for displaying
> - modify `blog_app/models.py` for redirection to `post_detail.html`

> ### Update post - LoginRequiredMixin
> - In `blog_app/views.py`
> - In `blog_app/urls.py`

> ### delete post
> - In `blog_app/views.py`
> - In `blog_app/urls.py`
> - Create `post_confirm_delete.html` in `blog_app/templates` folder
> - In `blog_app/views.py`
> - modify `post-detail.html`

- Pagination

> ### import from json file

```
- CMD: python manage.py shell
import json
from blog_app.models import Post
with open('post.json') as f:
   posts_json = json.load(f)
for post in posts_json:
   post = Post(title=post['title'], content=post['content'],
author_id = post['user_id'])
   post.save()
```

> - CMD: python manage.py shell

```
>>> from django.core.paginator import Paginator
>>> posts = ['1', '2', '3', '4', '5']
>>> p = Paginator(posts, 2)
>>> p.num_pages
3
>>> for page in p.page_range:
...     print(page)
...
1
```

```
2
3
>>> p.page(1)
<Page 1 of 3>
>>> p.page(1).number
1
>>> p.page(1).object_list
['1', '2']
>>> p.page(1).has_previous()
False
>>> p.page(1).has_next()
True
>>> p.page(1).next_page_number()
2
```

> - In `blog_app/views.py`
>
> ```
> class PostListView(ListView):
>    model = Post
>    template_name = 'blog_app/home.html' #
> <app>/<model>_<viewtype>.html
>    context_object_name = 'posts'
>    ordering = ['-date_posted']
>    paginate_by = 3
> ```

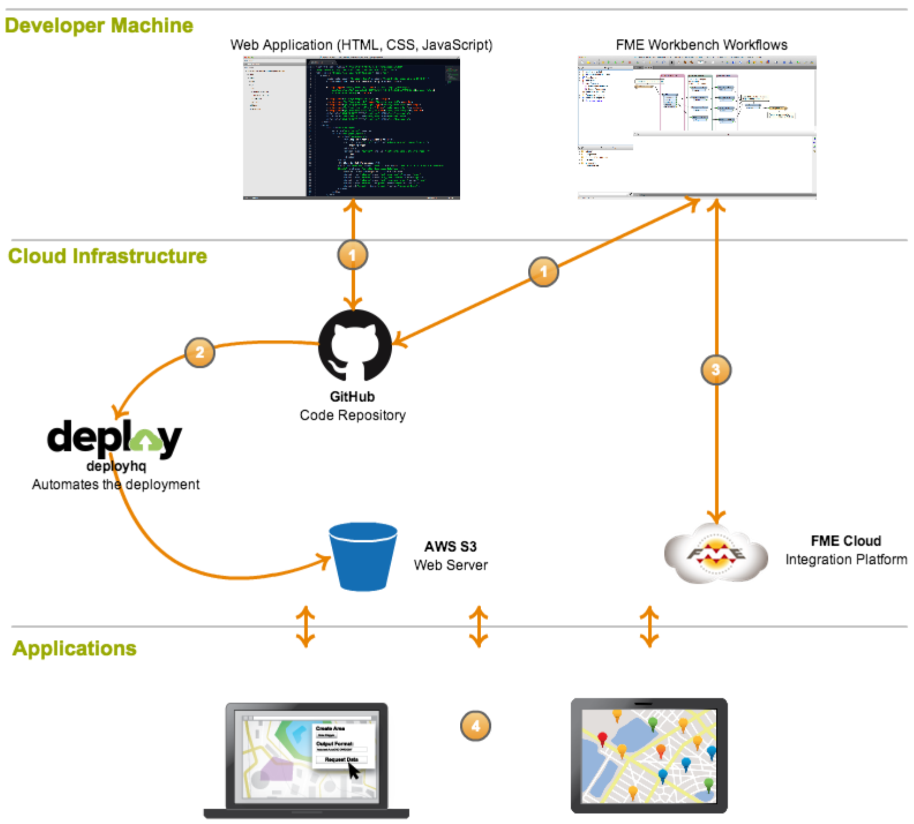
```
 {% if is_paginated %}

        {% if page_obj.has_previous %}
            <a class="btn btn-outline-info mb-4" href="?
page=1">First</a>
            <a class="btn btn-outline-info mb-4" href="?page={{
page_obj.previous_page_number }}">Previous</a>
        {% endif %}

        {% for num in page_obj.paginator.page_range %}
            {% if page_obj.number == num %}
                <a class="btn btn-info mb-4" href="?page={{ num
}}">{{ num }}</a>
            {% elif num > page_obj.number|add:'-3' and num <
page_obj.number|add:'3' %}
                <a class="btn btn-outline-info mb-4" href="?page=
{{ num }}">{{ num }}</a>
            {% endif %}
        {% endfor %}

        {% if page_obj.has_next %}
            <a class="btn btn-outline-info mb-4" href="?page={{
page_obj.next_page_number }}">Next</a>
            <a class="btn btn-outline-info mb-4" href="?page={{
page_obj.paginator.num_pages }}">Last</a>
        {% endif %}

    {% endif %}
```

- Enable HTTPS with SSL/TLS Certificate using Let's Encrypt

> Let's Encrypt Website

> `https://letsencrypt.org/getting-started/`
> -> click `Certbot`
> -> Apache, Ubuntu 18

Using AWS S3 for File Uploads



> ### Create AWS S3 Bucket
>
> AWS S3 Website
>
> - Create AWS S3 Bucket
>
> -> `django-learning-files`
>
> - Permession
> - CORS Configuration

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": [
      "Access-Control-Allow-Origin"
    ]
  }
]
```

Create New User in AWS S3

- IAM
- Add Users

-> django_user(Select AWS credential type: Access key - Programmatic access)
-> `Attach existing policies directly`
-> `AmazonS3FullAccess`
-> Access key ID + Secret access key

Link Django with AWS3 and Using .env to Store the Scecrte Variables

- `pip install boto3`
- `pip install django-storages`
- `pip install python-dotenv`

```
import os
from dotenv import load_dotenv
load_dotenv()
os.getenv('ENV_VAR')
```

- `setup .env`

-> AWS_STORAGE_BUCKET_NAME=*******
-> AWS_ACCESS_KEY_ID=********
-> AWS_SECRET_ACCESS_KEY=********

- Go to `settings.py`

```
-> import os
-> from dotenv import load_dotenv
-> load_dotenv()
-> INSTALLED_APPS = `[stroages]`
```

> -> AWS_STORAGE_BUCKET_NAME = os.getenv('AWS_STORAGE_BUCKET_NAME')
> -> AWS_ACCESS_KEY_ID = os.getenv('AWS_ACCESS_KEY_ID')
> -> AWS_SECRET_ACCESS_KEY = os.getenv('AWS_SECRET_ACCESS_KEY')
> -> AWS_S3_FILE_OVERWRITE = False
> -> AWS_DEFAULT_ACL = None
> -> DEFAULT_FILE_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

> - go to `user_app/models.py`
>
> -> # can not use below code due to AWS S3 for resizing images

> - upload images to the AWS S3 BUCKET

> ### Upload and Download files to AWS S3
>
> Reference: Upload and Download files from AWS S3 Bucket using python

```
# .ENV VARS CONFIG
load_dotenv()
aws_bucket_name = os.getenv('AWS_STORAGE_BUCKET_NAME')
aws_access_key_id = os.getenv('AWS_ACCESS_KEY_ID')
aws_secret_access_key= os.getenv('AWS_SECRET_ACCESS_KEY')

# S3 BUCKET CONFIG
s3 = boto3.resource("s3")
my_bucket = s3.Bucket(aws_bucket_name)
my_bucket.upload_file(Key='index.html',
Filename='./index.html')
my_bucket.download_file(Key='index.html',
Filename='./index.html')
```

Django Deployment Checklist

Deploy Preparation

> - `pip install gunicorn`
>
> -> CMD: gunicorn Django_Blog_Project.wsgi:application --bind 127.0.0.1:800

> - pipenv install waitress
>
> -> CMD: waitress-serve --listen=127.0.0.1:8000 Django_Blog_Project.wsgi:application

> - `pip install whitenoise`

> - Create `Procfile` in root directory `Django_Blog_Project`

> -> web: gunicorn Django_Blog_Project.wsgi --log-file -

> - Create `runtime.txt` in root directory `Django_Blog_Project`
>
> -> CMD: python --version
> -> put `python-3.10.8` into `runtime.txt`

> - **go to `settings.py`**

> -> DEBUG = (os.getenv('DEBUG_VALUE') == 'True')

> -> ALLOWED_HOSTS = ['*']

> -> STATIC_ROOT = os.path.join(BADE_DIR, 'staticfiles')
> -> python manage.py collectstatic

> -> MIDDLEWARE = [
> "django.middleware.security.SecurityMiddleware",
> "whitenoise.middleware.WhiteNoiseMiddleware",
>]

> - Create `Dockerfile`
>
> ```
> FROM python:3.10.8-slim-buster
> WORKDIR /app
> COPY ./Django_Blog_Project ./
>
> RUN pip install --upgrade pip --no-cache-dir
>
> RUN pip install -r /app/requirements.txt --no-cache-dir
>
> CMD ["python", "manage.py", "runserver", "127.0.0.1:8000"]
> CMD ["waitress-serve", "--listen=127.0.0.1:8000",
> "Django_Blog_Project.wsgi:application"]
> CMD ["gunicorn" "Django_Blog_Project.wsgi:application", "--
> bind", "0.0.0.0:8000"]
> ```

-- Memo End --