Neuresta

2021

Group 25

Jainendra Prakash

Shreyas MS

Ashar Siddiqui

Nishchay verma

Kavadi Sumanth kumar

Overview

There has been a tremendous amount of growth in the field of digital imaging and image stylization with the advancements in technology over the past few years. With this advancements in technology, we can now generate stylized images using a base image and a style image via a web application. Nowadays there is a lot of demand for image filters and image styling as the social media platforms have greatly increased their reach. So, in this project we have made a Neural Style Transfer WebApp which is easy to use and can generate a lot of stylized images for the user.

Goals

1. To enable the user to generate stylized images using the original image and a style image.
2. To help the user style his image using already existing stylized images from the gallery.
3. To help the user to store his images in his profile gallery.

Technology used

This website is made using :

- Django as the backend of the webapp(Python).
- Html, CSS javascript and Bootstrap in frontend.
- We have used SVG images to load faster web pages.
- Neural style transfer technique.
- Static pages are used to load faster.
- Pytorch is used for the Neural Style Transfer algorithm.

Specifications

- Home page contains some sample screenshots of the functioning website.
- The About page contains a short description of how the website works.
- Sign up page to register the user.
- Login page to login the already registered users.
- Image Upload page to upload two images one original image and one style image.

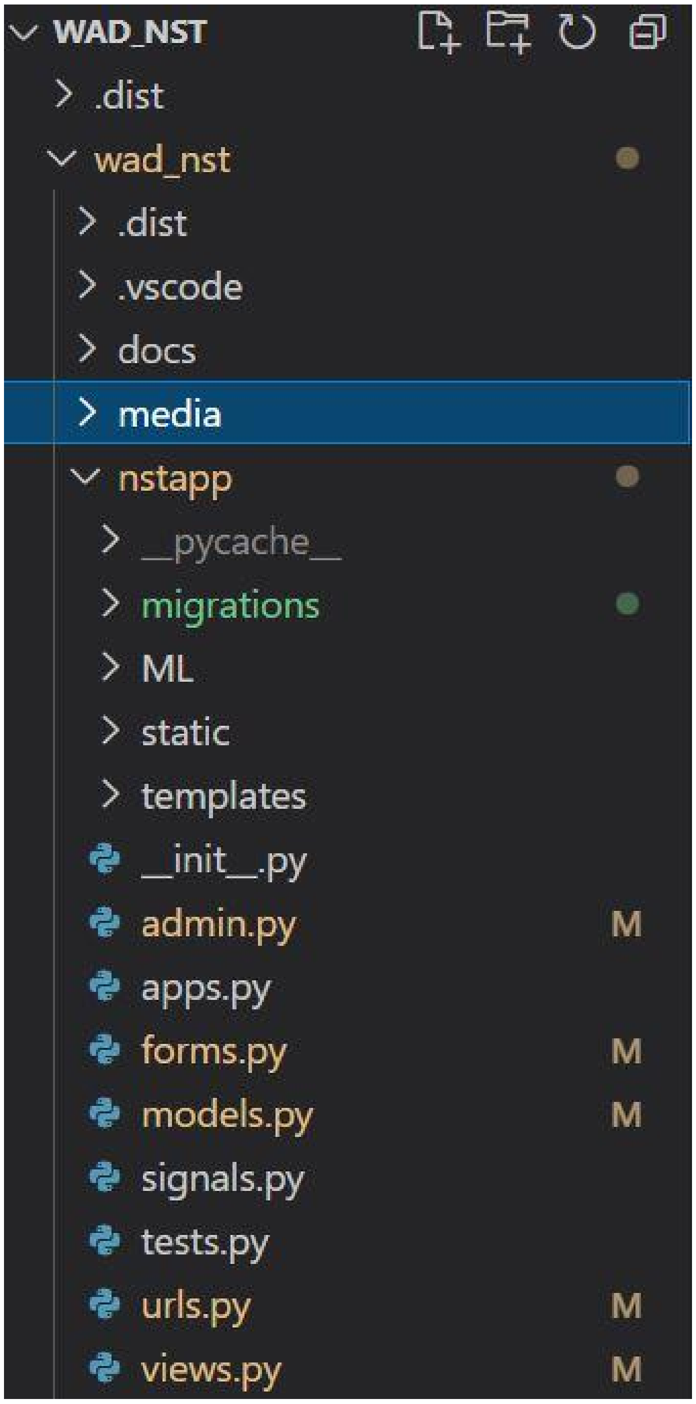
- A Gallery to show the final stylized images.
- A Profile page where the logged in user's data is shown.
- Profile update to update the user information.
- A Feedback page, if the user wants to give feedback about the website.

Abstract

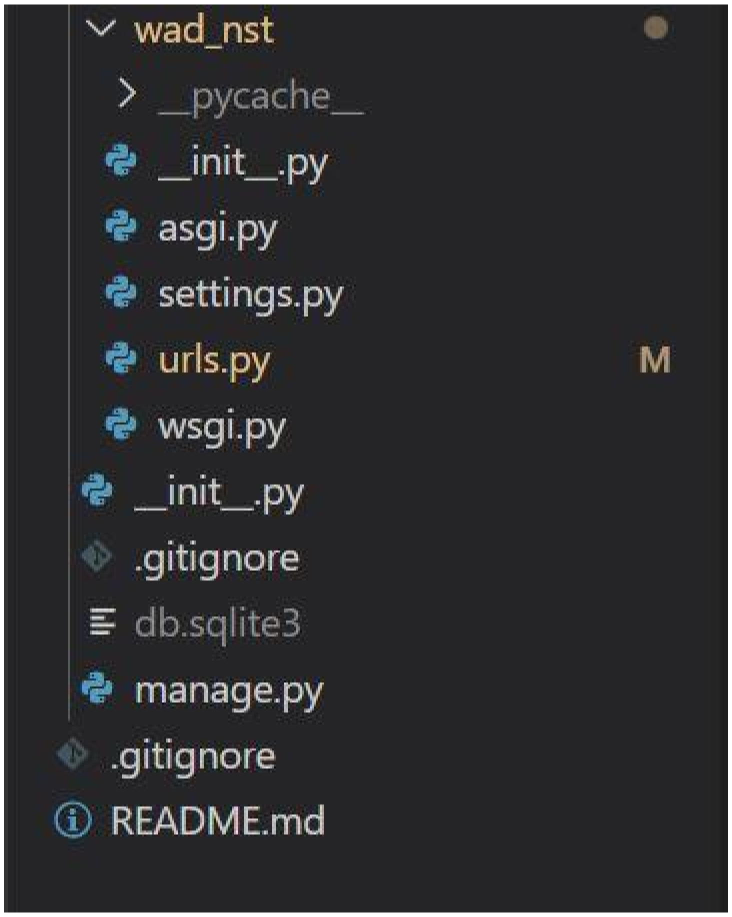
Neuresta is a website, where one can input two images, one original image and one style image to output a stylized image. In other worlds, we can get a type of image which has been made using an image filter. Here, we use the Neural Style Transfer technique to generate the stylized image. For our website we have used HTML, CSS, JavaScript for the frontend part and the backend runs on Django. The NST algorithm uses PyTorch. So, in Neuresta we give the user to style his or her image and the option to store the images in their own private gallery provided that the user is already logged in. In other words, our website is made for the user with the purpose of digital imaging and styling with an option to store their images.

Next, we would like to show all the components of our Neuresta website one by one and all the details of the web pages which we have in our website and how we would like the user to run our website or even how we can host the website.

All files and subfolders of our web app.



```
WAD_NST
  > .dist
  v wad_nst
    > .dist
    > .vscode
    > docs
    > media
    v nstapp
      > __pycache__
      > migrations
      > ML
      > static
      > templates
        __init__.py
        admin.py        M
        apps.py
        forms.py        M
        models.py       M
        signals.py
        tests.py
        urls.py         M
        views.py        M
```

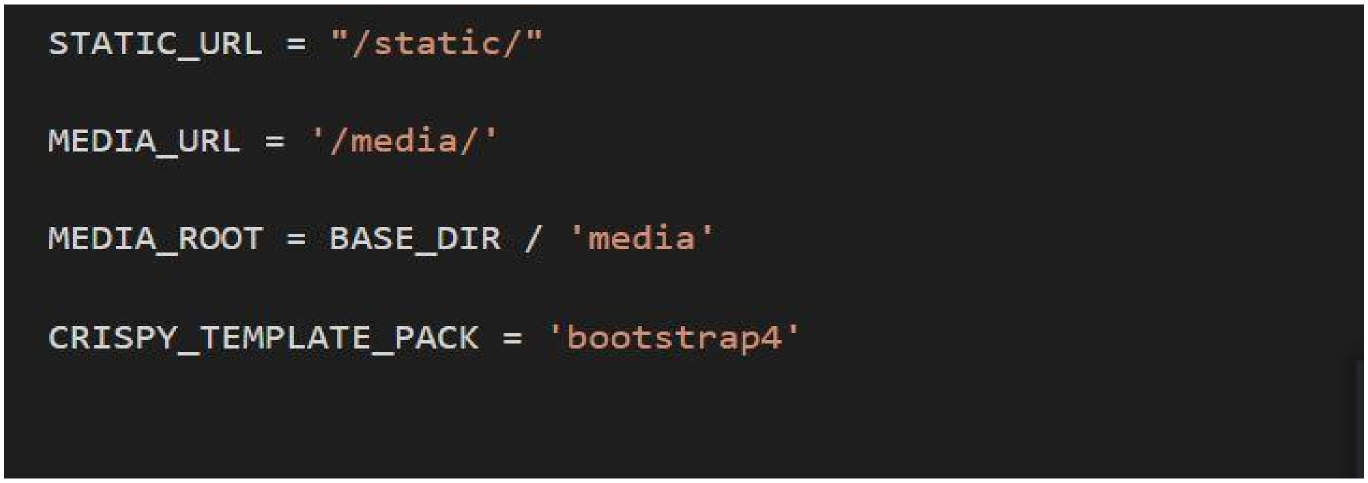


```
wad_nst
  __pycache__
  __init__.py
  asgi.py
  settings.py
  urls.py          M
  wsgi.py
__init__.py
.gitignore
db.sqlite3
manage.py
.gitignore
README.md
```

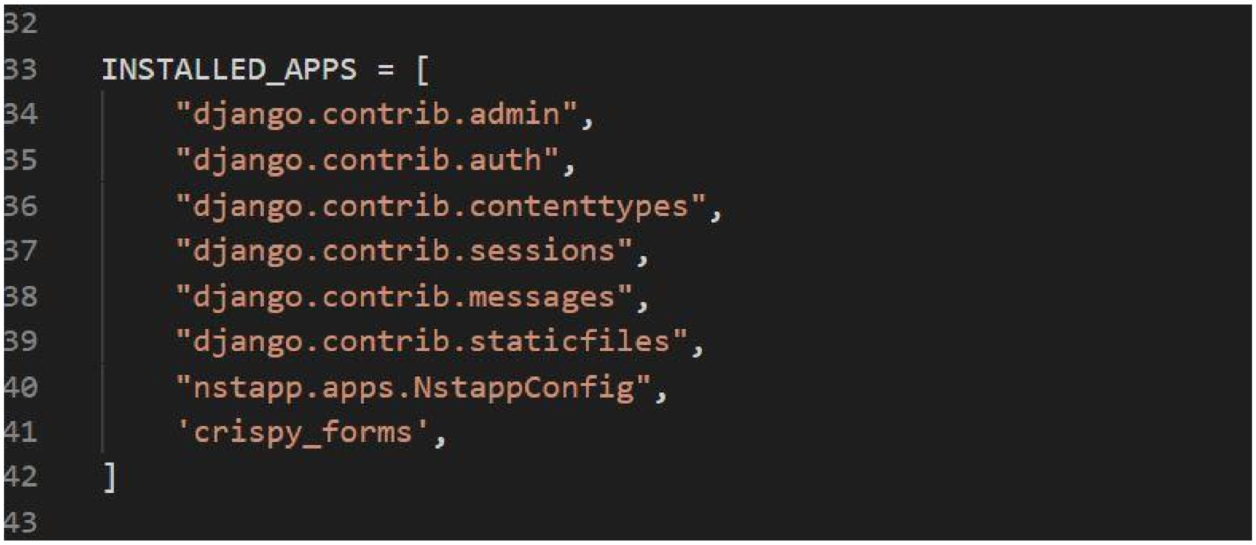
Run the website on your local host.

```
$ python manage.py runserver
```

Some changes made in settings.py for app installation and other routes.

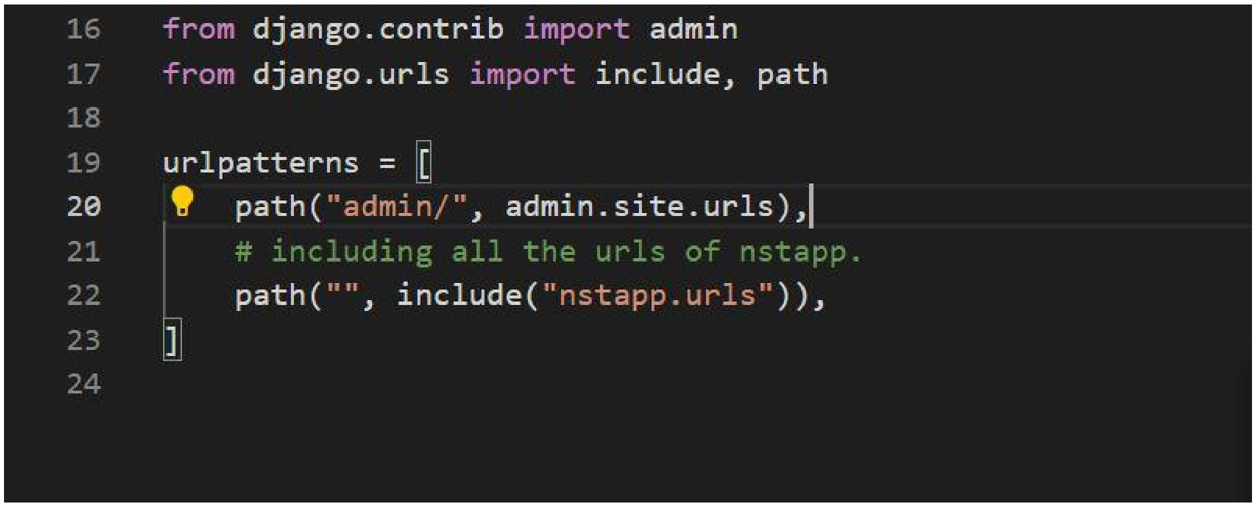


```
STATIC_URL = "/static/"

MEDIA_URL = '/media/'

MEDIA_ROOT = BASE_DIR / 'media'

CRISPY_TEMPLATE_PACK = 'bootstrap4'
```



```python
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "nstapp.apps.NstappConfig",
    'crispy_forms',
]
```

Urls.py of our web app main:



```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("admin/", admin.site.urls),
    # including all the urls of nstapp.
    path("", include("nstapp.urls")),
]
```

Here, we add an import for **django.urls.include** and insert an **include()** in the **urlpatterns** list.

Urls.py

```
from django.urls import path

from django.conf import settings

from django.conf.urls.static import static


from . import views


urlpatterns = [


    # Authentication of user links

    path("login/", views.loginuser, name="loginuser"),

    path("logout/", views.logoutuser, name="logoutuser"),

    path("signup/", views.signupuser, name="signupuser"),




    # Features of website links

    path("", views.home, name="home"),

    path("about/", views.about, name="about"),

    path("profile/", views.profile, name="profile"),

    path("gallery/", views.gallery, name="gallery"),

    path('feedback/',views.feedback,name = 'feedback'),

    path('profileupdate/',views.profileUpdate,name =
'profileUpdate'),
    path('upload/',views.imageupload,name = 'imageupload'),

]


# Static media links.

urlpatterns += static(settings.MEDIA_URL,document_root =
settings.MEDIA_ROOT)
```

Views.py

```
import os

import sys


from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.forms import AuthenticationForm,
UserCreationForm

from django.contrib.auth.models import User

from django.core.files.storage import FileSystemStorage

from django.db import IntegrityError

from django.shortcuts import redirect, render

from .forms import UserRegistrationForm, FeedbackForm,
UserUpdateForm, ProfileUpdateForm, ImageForm

from django.contrib.auth.decorators import login_required

from django.conf import settings

from pathlib import Path


PACKAGE_PARENT = ".."

SCRIPT_DIR = os.path.dirname(

    os.path.realpath(os.path.join(os.getcwd(),
os.path.expanduser(__file__)))
)
sys.path.append(os.path.normpath(os.path.join(SCRIPT_DIR,
PACKAGE_PARENT)))


from nstapp.ML.nst_run import run
```

Importing necessary files and classes required for our webpage some of them are inbuilt functions of django while others we have imported from our model and forms and our ML modules.

First we will walk you through UserCreation, UserAuthentication system.

Sign Up:

Signup function is to register the user into the website for that We are using django inbuilt model User and added extra field email in USerCreationForm and Created a new Form

As User RegistrationForm in forms.py.

When we get a request.POST method we are checking for some variables and afterwards we are the form into the database.

Returns

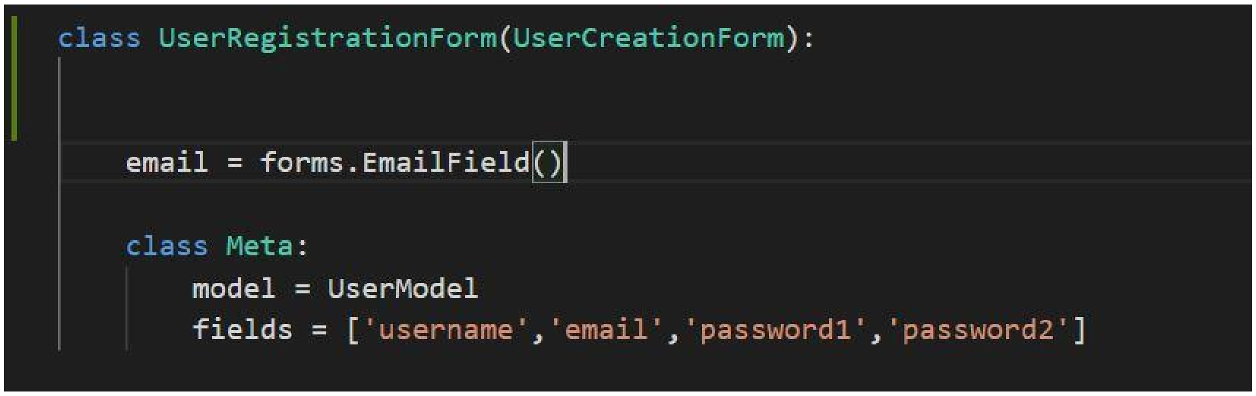
After success we are redirected to the homepage.

```
def signupuser(request):


    if request.method == "GET":
        return render(request, "signupuser.html", {"forms":
UserRegistrationForm()})
    else:
        if request.POST["password1"] == request.POST["password2"]:
            try:
                user = User.objects.create_user(
                    request.POST["username"],
                    password=request.POST["password1"],
                    email=request.POST["email"],
                )
                user.save()
                # When user signed in redirect to new url.
                login(request, user)
                return redirect("home")

            except IntegrityError:
                return render(
                    request,
                    "signupuser.html",
                    {
                        "forms": UserRegistrationForm(),
                        "error": "Username is already taken try
other!",
                    },
                )

        else:
            # Tell the use user password didn't match.
            return render(
                request,
                "signupuser.html",
                {"forms": UserCreationForm(), "error": "Password did
not match"},
            )
```

Next, we had built the new form UserRegistrationForm from the UserCreationForm function for the model user so that we can add

an extra field email. Necessary imports.

```
from django import forms
from django.contrib.auth.models import User as UserModel
from django.contrib.auth.forms import UserCreationForm
```



```
class UserRegistrationForm(UserCreationForm):


    email = forms.EmailField()

    class Meta:
        model = UserModel
        fields = ['username','email','password1','password2']
```

Login :

Logging in the user into the website means that we are using django for AuthenticationForm() when we get the POST method we authenticate the user from data we have in the database. If it doesn't match, we return an error. Then redirect it to the home page.

request :

Httprequest Django creates an HttpRequest object that contains metadata about the request. Then Django loads the appropriate view, passing the HttpRequest as the first argument to the view function. Each view is responsible for returning an HttpResponse object.

Returns:

If it's a get method e return form. Else if a post method and no error if redirected to the home page or else we return error.

```
def loginuser(request):

    if request.method == "GET":
        return render(request, "loginuser.html", {"forms": AuthenticationForm()})
    else:
        user = authenticate(
            request, username=request.POST["username"], password=request.POST["password"]
        )
        if user is None:
            return render(
                request,
                "loginuser.html",
                {"forms": AuthenticationForm(), "error": "Username and password did not match"},
            )
        else:
            login(request, user)
            return redirect("home")
```

Logout:

Logging out the user using django inbuilt function logout()

Returns

After logging out we are redirecting the user to the home page.

```
@login_required
def logoutuser(request):


    logout(request)
    return redirect("home")
```

@login_required means, we need to login to visit this page, once user will login, user will be redirected to home page.

Features of the Web pages:

Home page:

```
def home(request):
    return render(request, "home.html")
```

Home page is the showcase for the reviews, feedback and functioning of the website. It contains some nice animations which are made using HTML CSS JS and SVG images.

Home function will return the home.html template.

About page:

```
def about(request):
    return render(request, "about.html")
```

About page, is a simple static page which will show samples of output of websites and how the website works.

This function about will return about.html

Gallery page:

```
def gallery(request):
    return render(request, "gallery.html")
```

This page will show users images, which are stylized using Neuresta.

This function (gallery) will return this gallery.html page.

Profile page:

```
@login_required
def profile(request):
    return render(request, "profile.html")
```

Profile function will be visible if the user will login, and will return a profile.html page.

Profile page will contain the users personal information and their saved images.

Profile Update Page:

Here, we update the user profile but we keep the user data filled in it so that they have to edit only those parts that they wish to change. Using two forms UserUpdateForm and ProfileUpdateForm as there is no field of image in User model so we can't do it directly.

Redirecting it to the Profile page:

```
@login_required
def profileUpdate(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()
            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, "profileUpdate.html", context, {"error":
"Bad Data ! Try Again"}
            )

    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, "profileUpdate.html", context)
```

Creating a new model name Profile for our profile page that is having Foreign key constraint with User model. And another image field for profile pics that will be stored in profile_pics subfolder inside media if the upload or else default.jpg pic.

```
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"
```

UserUpdateForm and ProfileUpdateForm these two forms were created serve the purpose of profile update form.

```
class UserUpdateForm(ModelForm):
    email = forms.EmailField()

    class Meta:
        model = UserModel
        fields = ['username','email']

class ProfileUpdateForm(ModelForm):
    class Meta:
        model = ProfileModel
        fields = ['image']
```

Feedback Page:

A Feedback Form where users can post their views experience and suggestions for the admin. Similar to the login page here we have to create a model and the using forms we can take input from the user and store in the database.

Returns

If it's a get method e return form. Else if a post method and no error if redirected to the home page or else we return error.

```
@login_required
def feedback(request):


    if request.method == "GET":
        return render(request, "feedback.html", {"form":
FeedbackForm()})
    else:
        try:
            form = FeedbackForm(request.POST)  # Put all the data we
get from webpage
            newtodo = form.save()  # Saving the value.
            return redirect("home")
        except ValueError:
            return render(
                request, "feedback.html", {"form": FeedbackForm(),
"error": "Bad Data Try Again !"}
            )
```

Model of feedback form:

Creating a new model name Feedback to get the views of users about our website. We are storing their mail id for their reference. Created will automatically store the time feedback posted.

```
class Feedback(models.Model):
    email = models.EmailField(max_length=254)
    title = models.CharField(max_length=100)
    feedback = models.TextField(blank=True)
    created = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.title
```

Image Upload Form:

Image upload app to take user input two images and run it through the ML algorithm and generate the new styled image. For the run function

we need the address of images that we are collecting in path1,path2

and path3. For the simplicity sake as of now we fix the output image name.

Returns:

If it’s a get method return form,else if it is a post method and no error if redirect the user to image Generated page where user can see the image generated or else we return an error.

```
def imageupload(request):

    if request.method == "GET":
        return render(request, "imageupload.html", {"form":
ImageForm()})
    else:
        try:
            BASE_DIR = Path(__file__).resolve().parent.parent
            form = ImageForm(request.POST, request.FILES)   # Put all
the data we get from webpage
            newtodo = form.save()   # Saving the value.
            imageName1 = request.FILES["Content"].name
            imageName2 = request.FILES["Style"].name
            imageName3 = "abaa.jpg"
            path1 = BASE_DIR / "media" / "style" / imageName1
            path2 = BASE_DIR / "media" / "base" / imageName2

            path3 = BASE_DIR / "media" / "generated" / imageName3


            run(path2, path1, path3)


            return render(

                request,

                "about.html",

                {

                    "path1": path1,

                    "path2": path2,

                    "path3": path3,

                },

            )


        except ValueError:

            return render(

                request, "imageupload.html", {"form": ImageForm(),
"error": "Bad Data Try Again !"}

            )
```

```
class ImageForm(ModelForm):

    class Meta :

        model = ImageModel

        fields = ['image1','image2']
```

Created a model name Image to get a user image and feed it to the ML model and display the output. And storing the images in the media folder in their respective subfolder.

```
class Image(models.Model):


    image1 = models.ImageField(upload_to="style")

    image2 = models.ImageField(upload_to="base")

    image3 = models.ImageField(upload_to="generated")
```

TESTING

Testing is a process of executing a program with the interest of finding an error. A good test is one that has a high probability of finding the yet undiscovered error. Testing should systematically uncover different classes of errors in a minimum amount of time with a minimum amount of efforts. Two classes of inputs are provided to test the process

A software configuration that includes a software requirement specification, a design specification and source code.

A software configuration that includes a test plan and procedure, any testing tool and test cases and their expected results.

Testing is divided into several distinct operations:

1. **Unit Testing:** Unit test comprises a set of tests performed by an individual program prior to the integration of the unit into a large system. A program unit is usually the smallest free functioning part of the whole system.

Module unit testing should be as exhaustive as possible to ensure that each representation handled by each module has been tested. All the units that makeup the system must be tested independently to ensure that they work as required.

During unit testing some errors were raised and all of them were rectified and handled well. The result was quite satisfactory and it worked well.

2. Integration Testing: Integration testing is a system technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design. Bottom–up integration consists of unit tests followed by testing of the entire system. A sub–system consists of several modules that communicate with other defined interfaces.

The system was done for integration testing. All the modules were tested for their compatibility with other modules. Their test was almost successful. All the modules coexisted very well, with almost no bugs. All the modules were encapsulated very well so as to not hamper the execution of other modules.

3. Validation Testing: After validation testing, software is completely assembled as a package, interfacing errors that have been uncovered and corrected and the final series of software tests; the validation test begins. Steps taken during software design and testing can greatly improve the probability of successful integration in the larger system. System testing is actually a series of different tests whose primary purpose is to fully exercise the computer–based system.

4. Recovery Testing: It is a system that forces the software to fail in a variety of ways and verifies that the recovery is properly performed.

5. Security Testing: It attempts to verify that protection mechanisms built into a system will in fact protect it from improper penetration. The system's security must of course be tested from vulnerability from frontal attack.

6. Stress Testing: Stress tools are designed to confront programs with abnormal situations. Stress testing executes a system in a manner that demands resources in abnormal quantity and volume.

7. Black Box Testing: Black box testing is done to find out the following information as shown in below:

1. Incorrect or missing functions.
2. Interface errors
3. Errors or database access
4. Performance error
5. Termination error

The mentioned testing is carried out successfully for this application according to the user's requirement specification

8. Test Data Output: After preparing test data, the system under study is tested using the test data. While testing the system using test data, errors are again uncovered and corrected by using above testing and corrections are also noted for future use.

CONCLUSION

Our project is only a humble venture to satisfy the needs in an institution. Several user friendly coding have also been adopted. This package shall prove to be a powerful package in satisfying all the requirements of the organization.

The objective of software planning is to provide a framework that enables the manager to make reasonable estimates made within a limited time frame at the beginning of the software project and should be updated regularly as the project progresses.