**SOURCE CODE FOR PHASE 3 - TRENDYFIT**

**CONTENTS**

**LIST OF CONTENTS PAGE NO**

**1**[**.**](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.fzahbmza80ps) **SOURCE CODE 1**

1[.1.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.lh1bjxf81rog) Admin Views.py 1

1.2. Admin Models.py 3

1.3. Designer Views.py 4

1.4. Designer.html 6

1.5. Users Views.py 9

1.6. Users Models.py 11

**2**[**.**](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.2pv3oupaurdx) **CODE SNIPPETS 12**

2[.1.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Home page view 13

2[.2.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Designer view 13

2[.3.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) User view 13

2[.4.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) plan view 14

2[.5.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription View 14

2[.6.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Plan Model 14

2[.7.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 15

2[.8.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 15

2[.9.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 16

2[.10.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 16

2[.11.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 17

2[.12.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 18

2[.13.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 18

2[.14.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 19

2[.15.](https://docs.google.com/document/d/1kJZktaO0pPRdo00rmaDJq1UpTCWYdKsT6pmtnlOFwHk/edit#heading=h.4vjmtlf43ja7) Subscription Model 19

1. **SOURCE CODE:**

**1.1. Admin views.py**

from .forms import ConnectionForm, CustomNotificationForm, PlanForm

from .models import Plan, Subscription, CustomNotification, Connection

from user.models import User

from django.http.response import JsonResponse

from django.shortcuts import redirect, render

from django.views import View

from django.contrib.auth.mixins import LoginRequiredMixin

from django.db.models import F, Count

from channels.layers import get\_channel\_layer

from asgiref.sync import async\_to\_sync

import json

**# home page view**

class HomeView(LoginRequiredMixin, View):

template\_name = "main/home.html"

def get\_plan\_list(self):

return Plan.objects.all()

def get\_active\_user(self, active):

return User.objects.filter(is\_active=active).count()

def get\_subscription(self,active):

return Subscription.objects.filter(is\_active=active).count()

def get\_plans(self):

return Plan.objects.filter().count()

def get(self, request, \*args, \*\*kwargs):

if request.user.is\_superuser:

return render(

request,

self.template\_name,

{

"active\_user": self.get\_active\_user(True),

"non\_active\_user": self.get\_active\_user(False),

'subscription\_active': self.get\_subscription(True),

'subscription\_not\_active' : self.get\_subscription(True),

'number\_of\_plans':self.get\_plans()

},

)

return render(request, self.template\_name, {"plans": self.get\_plan\_list()})

**# designer view**

class DesignerView(LoginRequiredMixin, View):

template\_name = "main/designer.html"

def get\_list(self):

return User.objects.filter(user\_type="DS").exclude(is\_superuser=True)

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"designers": self.get\_list()})

**# user view**

class UserView(LoginRequiredMixin, View):

template\_name = "main/user.html"

def get\_list(self):

return User.objects.filter(user\_type="NU").exclude(is\_superuser=True)

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"users": self.get\_list()})

**# plan view**

class PlanView(LoginRequiredMixin, View):

template\_name = "main/plan.html"

def get\_list(self):

return Plan.objects.all()

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"plans": self.get\_list()})

class PlanAddView(LoginRequiredMixin, View):

template\_name = "main/add\_plan.html"

def get(self, request, \*args, \*\*kwargs):

form = PlanForm()

return render(request, self.template\_name, {"form": form})

def post(self, request, \*args, \*\*kwargs):

try:

form = PlanForm(request.POST)

if form.is\_valid():

form.save()

return redirect("plan")

return render(request, self.template\_name, {"form": form})

except Exception as e:

return render(request, self.template\_name, {"form": form})

**#Subscription View**

class SubscriptionView(LoginRequiredMixin, View):

template\_name = "main/subscription.html"

def get\_list(self):

return Subscription.objects.all()

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"subscriptions": self.get\_list()})

**Description:** Views are Python functions or classes that receive a web request and deliver a web response in the Django framework, as shown in the following source code from Django views.py files. The possible responses can be a plain HTTP response, an HTML template response, or an HTTP redirect response that sends the user to another page

**1.2. Admin Models.py**

from enum import unique

from django.db.models import BooleanField, DateTimeField, ImageField, TextField

from django.db.models.fields import TextField

from user.models import User

from django.db import models

from django.db.models import CASCADE

from common.models import BasicBaseModel

from django.db.models import CharField, IntegerField

**# importing DateTime module**

from datetime import datetime, timedelta

request\_status\_choices = [('Pending',"PD"),("Accepted","AD")]

class Plan(BasicBaseModel):

name = CharField(max\_length=30)

number\_of\_days = IntegerField(null=False, blank=False)

price = IntegerField(null=False, blank=False)

class Subscription(BasicBaseModel):

user = models.ForeignKey(User, on\_delete=CASCADE)

plan = models.ForeignKey(Plan, on\_delete=CASCADE)

start\_timestamp = models.DateTimeField(auto\_now\_add=True)

end\_timestamp = DateTimeField(blank=False, null=True)

payment\_method = CharField(max\_length=30)

is\_active = BooleanField(default=False, null=False, blank=False)

def save(self, \*args, \*\*kwargs):

self.end\_timestamp = datetime.now() + timedelta(days=self.plan.number\_of\_days)

super(Subscription, self).save(\*args, \*\*kwargs)

**Description:** Models are Python objects that Django web applications use to access and manage data. Models describe the structure of stored data, including field types and, in some cases, their maximum size, default values, selection list options, documentation help text, and form label text. Models.py has classes named plan and subscription in the following code, where the plan keeps information such as plan name, number of days, and price for each plan. In the case of the subscription class, it holds information such as the user, the type of plan selected, as well as the start and end dates of the subscription.

**1.3. Designer views.py**

class NotificationView(LoginRequiredMixin, View):

template\_name = "main/notification.html"

def get\_list(self, user):

return CustomNotification.objects.select\_related("from\_user", "to\_user").filter(

to\_user=user

)

def get(self, request, \*args, \*\*kwargs):

print(request.user)

print(self.get\_list(request.user))

return render(

request, self.template\_name, {"notifications": self.get\_list(request.user)}

)

def post(self, request):

try:

data = request.POST.dict()

if data["type"] == "connection":

data["from\_user"] = request.user

form = CustomNotificationForm(data=data)

if form.is\_valid():

form.save()

connec\_obj = Connection.objects.create(

from\_user=data["from\_user"], to\_user\_id=data["to\_user"]

)

channel\_layer = get\_channel\_layer()

channel = f"notification\_{str(request.user.id)}"

async\_to\_sync(channel\_layer.group\_send)(

channel,

{

"type": "notify",

"command": "new\_notification",

"from\_user": connec\_obj.from\_user.email,

"to\_user": connec\_obj.to\_user.first\_name,

"message": data["message"],

},

)

return JsonResponse({"message": ["Success"]})

return JsonResponse({"error": [form.errors]}, status=400)

return JsonResponse({"error": ["command not found"]}, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

class ConnectionUpdateView(LoginRequiredMixin, View):

def get(self, request):

data = Connection.objects.filter(

Q(from\_user=request.user, status="Accepted")

| Q(to\_user=request.user, status="Accepted")

).values(

["from\_user\_email", "from\_user\_\_image", "to\_user\_\_image", "to\_user\_\_email"]

)

return JsonResponse({"data": list(data)})

def post(self, request, id, \*args, \*\*kwargs):

try:

form = ConnectionForm(

request.POST,

instance=Connection.objects.get(from\_user\_id=id, to\_user=request.user),

)

if form.is\_valid():

form.save()

CustomNotification.objects.filter(

from\_user\_id=id, to\_user=request.user

).update(seen=True)

return redirect("notification")

return JsonResponse({"error": [form.errors]}, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

class ConnectionView(LoginRequiredMixin, View):

def get(self, request):

data = Connection.objects.filter(

Q(from\_user=request.user, status="Accepted")

| Q(to\_user=request.user, status="Accepted")

).values(

"from\_user\_\_email", "from\_user\_\_image", "to\_user\_\_image", "to\_user\_\_email"

)

return JsonResponse({"data": list(data)})

class ChatView(LoginRequiredMixin, View):

template\_name = "main/chat.html"

def get(self, request, \*args, \*\*kwargs):

form = PlanForm()

return render(request, self.template\_name)

**Description:** Views are Python functions or classes that receive a web request and deliver a web response in the Django framework, as shown in the following source code from Django views.py files. The possible responses can be a plain HTTP response, an HTML template response, or an HTTP redirect response that sends the user to another page.

**1.4. Designer.html**

{% extends 'base\_layout/index.html' %}

{% load static %}

{% load main\_tags %}

{% block body %}

<div class='designer-outer-body mt-5'>

<div class='container-fluid row mx-auto'>

{% for designer in designers %}

<div class="col col-md-6 col-xl-4 mb-4 mb-lg-0">

<div class="card mb-3" style="border-radius: .5rem;">

<div class="row g-0">

<div class="col-md-4 gradient-custom text-center text-white"

style="border-top-left-radius: .5rem; border-bottom-left-radius: .5rem;">

<img src={% if designer.imgae %}{{designer.image.url}}{% else %}"static/img/default.png"{% endif %} class="profile-img-circular mt-5" />

<h6 class='mt-2'>{{designer.first\_name}} {{designer.last\_name}}</h6>

<p>{{designer.user\_type}}</p>

{% if designer.is\_active %}

<button class='btn btn-danger designer-deactivate'

user-url={% url 'user-update' designer.id %}>Deactivate</button>

{% else %}

<button class='btn btn-success designer-activate'

user-url={% url 'user-update' designer.id %}>Activate</button>

{% endif %}

</div>

<div class="col-md-8">

<div class="card-body p-4">

<h6>Information</h6>

<hr class="mt-0 mb-2">

<div class="row pt-1">

<div class="col-4 mb-2">

<h6>Email</h6>

<p class="text-muted"><small>{{designer.email}}</small></p>

</div>

<div class="col-4 mb-2">

<h6>Date Joined</h6>

<p class="text-muted">{{designer.date\_joined|date:"Y-m-d"}} </p>

</div>

<div class="col-4 mb-2">

<h6>Date Joined</h6>

<p class="text-muted">{{designer.last\_login|date:"Y-m-d"}}</p>

</div>

</div>

<h6>Subscription Info</h6>

<hr class="mt-0">

<div class="row pt-1">

<div class="col-4 ">

<h6>Plan Name</h6>

<p class="text-muted "> <small>{{designer.subscription\_set.all|plan\_details}}

</small></p>

</div>

<div class="col-4 ">

<h6>Days Left</h6>

<p class="text-muted"><small>{{designer.subscription\_set.all|numRmDays}}</small></p>

</div>

<div class="col-4 ">

<h6>Plan Status</h6>

{% if designer.is\_active %}

<p class=" text-success">

Active

</p>

{% else %}

<p class="text-danger">

Deactive

</p>

{% endif %}

</div>

</div>

</div>

</div>

</div>

</div>

</div>

{% endfor %}

</div>

</div>

{% endblock body %}

{% block custom-script %}

<script src={% static 'js/designer.js' %}></script>

{% endblock custom-script %}

**Description:** The above code is an HTML template for our designer dashboard, which gives the designer access to four features: notifications, user views, and user chat, and also, check plans on the homepage.

**1.5. User views.py**

import urllib

import json

from .forms import PaymentForm, UserForm, SignUpForm

from .models import User

from main.models import Plan, Subscription

from user.custom\_permission import AdminRequiredJsonMixin

from django.views import View

from django.contrib.auth.views import LoginView

from django.http import JsonResponse

from django.views.generic.edit import FormView, CreateView

from django.shortcuts import redirect, render

**#Customer login view**

class CustomLoginView(LoginView):

template\_name = "user/login.html"

redirect\_authenticated\_user = True

**#user view**

class UserView(AdminRequiredJsonMixin, View):

def post(self, request, id, \*args, \*\*kwargs):

try:

user\_instance = User.objects.get(id=id)

form = UserForm(request.POST, instance=user\_instance)

if form.is\_valid():

form.save()

return JsonResponse({"message": ["success"]})

return JsonResponse(form.errors, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

**#signup view**

class SignUpView(FormView, CreateView):

form\_class = SignUpForm

template\_name = "user/signup.html"

success\_url = "/home"

def form\_valid(self, form):

if form.cleaned\_data.get("user\_type") == "DS":

self.success\_url = f"/payment/{form.cleaned\_data.get('email')}/"

return super(SignUpView, self).form\_valid(form)

**#payment view**

class PaymentView(FormView):

template\_name = "user/payment.html"

success\_url = "/home"

form\_class = PaymentForm

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['userEmail'] = self.user\_email

return context

def get(self,request,user\_email):

self.user\_email=user\_email

return super(PaymentView,self).get(request)

def post(self, request,user\_email,\*args, \*\*kwargs):

try:

plan = Plan.objects.get(id=request.POST["plan"])

user = User.objects.get(email=user\_email)

Subscription.objects.create(

user=user,

plan=plan,

payment\_method=request.POST["payment\_method"],

is\_active=True

)

except Exception as e:

pass

return redirect("home")

**Description:** Views are Python functions or classes that receive a web request and deliver a web response in the Django framework, as shown in the following source code from Django views.py files. The possible responses can be a plain HTTP response, an HTML template response, or an HTTP redirect response that sends the user to another page.

**1.6. User Models.py**

import uuid

from django.db import models

from django.db.models.fields import CharField, EmailField, UUIDField

from django.contrib.auth.models import AbstractUser, BaseUserManager

from django.utils.translation import ugettext\_lazy as \_

USERTYPE = (("DS", "Designer"), ("NU","Normal User"))

class UserManager(BaseUserManager):

"""Helps in creating a user

Args:

BaseUserManager ([type]): [description]

Raises:

ValueError: if the user status is not matched

Returns:

User: User object in django

"""

use\_in\_migrations = True

def \_create\_user(self, email: str, password: str, \*\*extra\_fields):

"""Create and save a User with given email and password.

Raises:

ValueError: if email is null

Returns:

User: User object in django

"""

if not email:

raise ValueError("Please provide a valid email")

user = self.model(email=self.normalize\_email(email), \*\*extra\_fields)

user.set\_password(password)

user.save(using=self.\_db)

return user

def create\_user(self, email: str, password: str = None, \*\*extra\_fields):

"""Create & save a normal User with given email and password

Returns:

User: User object in django

"""

extra\_fields.setdefault("is\_staff", False)

extra\_fields.setdefault("is\_superuser", False)

return self.\_create\_user(email, password, \*\*extra\_fields)

def create\_superuser(self, email: str, password: str, \*\*extra\_fields):

"""[summary]

Args:

email str: [description]

password str: [description]

Raises:

ValueError: [description]

ValueError: [description]

Returns:

User: User object in django

"""

extra\_fields.setdefault("is\_staff", True)

extra\_fields.setdefault("is\_superuser", True)

if extra\_fields.get("is\_staff") is not True:

raise ValueError("Superuser must have is\_staff True.")

if extra\_fields.get("is\_superuser") is not True:

raise ValueError("Superuser must have is\_superuser True.")

return self.\_create\_user(email, password, \*\*extra\_fields)

# Create your models here.

class User(AbstractUser):

username = None

id = UUIDField(primary\_key=True, default=uuid.uuid4, editable=False)

email = EmailField(\_("email"), max\_length=254, unique=True)

image = models.ImageField(upload\_to='profile\_picture',null=False,blank=False)

force\_password\_change = models.BooleanField(default=False)

user\_type = CharField(max\_length=2,choices=USERTYPE,default='NU',null=False,blank=False)

objects = UserManager()

USERNAME\_FIELD = "email"

REQUIRED\_FIELDS = ['first\_name','last\_name']

def \_\_str\_\_(self):

return self.email

class Meta:

ordering = ["id"]

**Description:** Models are Python objects that Django web applications use to access and manage data. Models describe the structure of stored data, including field types and, in some cases, their maximum size, default values, selection list options, documentation help text, and form label text. Models.py has classes named. User manager and user are classes in Models.py, and the user manager class provides functions like create user, which accepts email and password as parameters, as well as additional data, and the user class keeps all the user's fields, which store each user's information.

**2. CODE SNIPPETS:**

**2.1. Home page view**

class HomeView(LoginRequiredMixin, View):

template\_name = "main/home.html"

def get\_plan\_list(self):

return Plan.objects.all()

def get\_active\_user(self, active):

return User.objects.filter(is\_active=active).count()

def get\_subscription(self,active):

return Subscription.objects.filter(is\_active=active).count()

def get\_plans(self):

return Plan.objects.filter().count()

def get(self, request, \*args, \*\*kwargs):

if request.user.is\_superuser:

return render(

Request,

self.template\_name,

{

"active\_user": self.get\_active\_user(True),

"non\_active\_user": self.get\_active\_user(False),

'subscription\_active': self.get\_subscription(True),

'subscription\_not\_active' : self.get\_subscription(True),

'number\_of\_plans':self.get\_plans()

},

)

return render(request, self.template\_name, {"plans": self.get\_plan\_list()})

**Description:** We have a class named HomeView in the preceding code that renders the application's homepage. The functions in this class provide the activities that an admin can perform while visiting the home page, such as viewing plans, checking active users (which reveals if a user is active or not), and seeing subscriptions. View the list of the plans in the same way that the number of plans is displayed.

**2.2. Designer view**

class DesignerView(LoginRequiredMixin, View):

template\_name = "main/designer.html"

def get\_list(self):

return User.objects.filter(user\_type="DS").exclude(is\_superuser=True)

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"designers": self.get\_list()})

**Description:**

In the previous code, we have a class named DesignerView that renders Designer profiles that have been registered with our application. The functions in this class provide the tasks that an admin can perform while accessing this feature, such as browsing a list of registered designers, checking their details such as id, whether the designer is active or not, and the plan they have selected.

**2.3. User view**

class UserView(LoginRequiredMixin, View):

template\_name = "main/user.html"

def get\_list(self):

return User.objects.filter(user\_type="NU").exclude(is\_superuser=True)

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"users": self.get\_list()})

**Description:**

We have a class named UserView in the following code sample that renders user-profiles registered with our application. The functions in this class give actions that an admin can accomplish while using this feature, such as reading a list of registered users, checking their details like id, whether the designer is active or not, and the plan they have chosen.

**2.4. Plan view**

class PlanView(LoginRequiredMixin, View):

template\_name = "main/plan.html"

def get\_list(self):

return Plan.objects.all()

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"plans": self.get\_list()})

#**plan add view**

class PlanAddView(LoginRequiredMixin, View):

template\_name = "main/add\_plan.html"

def get(self, request, \*args, \*\*kwargs):

form = PlanForm()

return render(request, self.template\_name, {"form": form})

def post(self, request, \*args, \*\*kwargs):

try:

form = PlanForm(request.POST)

if form.is\_valid():

form.save()

return redirect("plan")

return render(request, self.template\_name, {"form": form})

except Exception as e:

return render(request, self.template\_name, {"form": form})

**Description:** In the following code sample, the classes named Planview and PlanAddView renders plans registered by users while registering with our application. The functions in this class describe actions that an admin can take while using this feature, such as reading a list of available plans for users, such as silver, gold, and diamond plans, each with a different validity period, such as 3, 6, or 12 months, and then viewing or adding any of these plans to the users who have registered with them.

**2.5. Subscription View**

class SubscriptionView(LoginRequiredMixin, View):

template\_name = "main/subscription.html"

def get\_list(self):

return Subscription.objects.all()

def get(self, request, \*args, \*\*kwargs):

return render(request, self.template\_name, {"subscriptions": self.get\_list()})

**Description:** The class SubscriptionView in the preceding code sample enables the administrator to view subscriptions in the applications. When an administrator requests to view a subscription, the programme accepts the request and searches the database for the subscription, which is then displayed on the dashboard.

**2.6. Plan model**

class Plan(BasicBaseModel):

name = CharField(max\_length=30)

number\_of\_days = IntegerField(null=False, blank=False)

price = IntegerField(null=False, blank=False)

**Description:** The plan class in the above code stores information about each plan, such as its name, number of days, and price. Users can choose from three different plans: silver, which has a three-month validity, gold, which has a six-month validity, and diamond, which has a 12-month validity.

**2.7. Subscription model**

class Subscription(BasicBaseModel):

user = models.ForeignKey(User, on\_delete=CASCADE)

plan = models.ForeignKey(Plan, on\_delete=CASCADE)

start\_timestamp = models.DateTimeField(auto\_now\_add=True)

end\_timestamp = DateTimeField(blank=False, null=True)

payment\_method = CharField(max\_length=30)

is\_active = BooleanField(default=False, null=False, blank=False)

def save(self, \*args, \*\*kwargs):

self.end\_timestamp = datetime.now() + timedelta(days=self.plan.number\_of\_days)

super(Subscription, self).save(\*args, \*\*kwargs)

**Description:** The subscription class in the following code contains information such as the userid, the plan type selected, and the subscription's start and end dates.

**2.8. Notification view**

class NotificationView(LoginRequiredMixin, View):

template\_name = "main/notification.html"

def get\_list(self, user):

return CustomNotification.objects.select\_related("from\_user", "to\_user").filter(

to\_user=user

)

def get(self, request, \*args, \*\*kwargs):

print(request.user)

print(self.get\_list(request.user))

return render(

request, self.template\_name, {"notifications": self.get\_list(request.user)}

)

def post(self, request):

try:

data = request.POST.dict()

if data["type"] == "connection":

data["from\_user"] = request.user

form = CustomNotificationForm(data=data)

if form.is\_valid():

form.save()

connec\_obj = Connection.objects.create(

from\_user=data["from\_user"], to\_user\_id=data["to\_user"]

)

channel\_layer = get\_channel\_layer()

channel = f"notification\_{str(request.user.id)}"

async\_to\_sync(channel\_layer.group\_send)(

channel,

{

"type": "notify",

"command": "new\_notification",

"from\_user": connec\_obj.from\_user.email,

"to\_user": connec\_obj.to\_user.first\_name,

"message": data["message"],

},

)

return JsonResponse({"message": ["Success"]})

return JsonResponse({"error": [form.errors]}, status=400)

return JsonResponse({"error": ["command not found"]}, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

**Description:** The Notification view class in the above code renders the application's notification page. Designers can check their notifications here, such as user connection requests or messages. As the user's id is displayed, the Designer may identify who the request is from and click to approve or reject it.

**2.9. ConnectionUpdate view**

**#user view**

class ConnectionUpdateView(LoginRequiredMixin, View):

def get(self, request):

data = Connection.objects.filter(

Q(from\_user=request.user, status="Accepted")

| Q(to\_user=request.user, status="Accepted")

).values(

["from\_user\_email", "from\_user\_\_image", "to\_user\_\_image", "to\_user\_\_email"]

)

return JsonResponse({"data": list(data)})

def post(self, request, id, \*args, \*\*kwargs):

try:

form = ConnectionForm(

request.POST,

instance=Connection.objects.get(from\_user\_id=id, to\_user=request.user),

)

if form.is\_valid():

form.save()

CustomNotification.objects.filter(

from\_user\_id=id, to\_user=request.user

).update(seen=True)

return redirect("notification")

return JsonResponse({"error": [form.errors]}, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

**Description:**

In the above code, we have a ConnectionupdateView class. Which has a function that updates the status of the user's connection requests, such as whether they are accepted or rejected, and sends the status of the connection request to the user as a notification.

**2.10. Connection view**

class ConnectionView(LoginRequiredMixin, View):

def get(self, request):

data = Connection.objects.filter(

Q(from\_user=request.user, status="Accepted")

| Q(to\_user=request.user, status="Accepted")

).values(

"from\_user\_\_email", "from\_user\_\_image", "to\_user\_\_image", "to\_user\_\_email"

)

return JsonResponse({"data": list(data)})

**Description:**

We have a class called Connection View in the following code snippet that displays the details of the user whose connection request has been accepted, and they may now interact with that user.

**2.11. Chat view**

class ChatView(LoginRequiredMixin, View):

template\_name = "main/chat.html"

def get(self, request, \*args, \*\*kwargs):

form = PlanForm()

return render(request, self.template\_name)

**Description:** The following code sample uses the class chat view to render a chat.html page where the Designer and the user can connect and interact. It also shows all users who have had their connection requests accepted, so Designers can choose whom they want to talk with.

**2.12. Customer login view**

**#Customer login view**

class CustomLoginView(LoginView):

template\_name = "user/login.html"

redirect\_authenticated\_user = True

**Description:** In the above code, we have a customer login view class that renders the application's login page. users may access their dashboards from here, and new users can sign up using the signup option.

**2.13. User view**

**#user view**

class UserView(AdminRequiredJsonMixin, View):

def post(self, request, id, \*args, \*\*kwargs):

try:

user\_instance = User.objects.get(id=id)

form = UserForm(request.POST, instance=user\_instance)

if form.is\_valid():

form.save()

return JsonResponse({"message": ["success"]})

return JsonResponse(form.errors, status=400)

except Exception as e:

return JsonResponse({"error": [e.\_\_str\_\_()]}, status=400)

**Description:**

We have a class called UserView in the previous code. Which has a function that requests the user's id from user form data; if found, it returns a status with the message success, and if not, it returns a JSON response with the status code 400 indicating that the operation failed.

**2.14. Signup view**

**#signup view**

class SignUpView(FormView, CreateView):

form\_class = SignUpForm

template\_name = "user/signup.html"

success\_url = "/home"

def form\_valid(self, form):

if form.cleaned\_data.get("user\_type") == "DS":

self.success\_url = f"/payment/{form.cleaned\_data.get('email')}/"

return super(SignUpView, self).form\_valid(form)

**Description:**

In the following code sample, we have a class named Signup View that renders the application's signup page, which displays a form and asks for information such as email address, password, and other personal information. The user gets returned to the home page after successfully enrolling for the application.

**2.15. Payment view**

**#payment view**

class PaymentView(FormView):

template\_name = "user/payment.html"

success\_url = "/home"

form\_class = PaymentForm

def get\_context\_data(self, \*\*kwargs):

context = super().get\_context\_data(\*\*kwargs)

context['userEmail'] = self.user\_email

return context

def get(self,request,user\_email):

self.user\_email=user\_email

return super(PaymentView,self).get(request)

def post(self, request,user\_email,\*args, \*\*kwargs):

try:

plan = Plan.objects.get(id=request.POST["plan"])

user = User.objects.get(email=user\_email)

Subscription.objects.create(

user=user,

plan=plan,

payment\_method=request.POST["payment\_method"],

is\_active=True

)

except Exception as e:

pass

return redirect("home")

**Description:** The class payment view in the following code sample will render a payment page where the user can approve payment for the subscription plan of their choice. The user will be taken back to the home page after making a successful payment.