**Code Inspection for Phase1 Development**

**User -> forms.py:**

forms.py under User module file contains Fuelxpress\_UserRegisterForm, Fuelxpress\_UserUpdateForm, Fuelxpress\_ProfileUpdateForm which extends Django built in forms for User registration, login and profile creation. Using the below logic, we are customizing the forms as per our requirement.

Meta classes will define form behavior and their relationships to respective DB model using the configuration settings provided.

class Fuelxpress\_UserRegisterForm(UserCreationForm):

FE\_email = forms.EmailField() #Email field

#Configure from metadata

class meta:

FE\_model = User

FE\_fields = ['username', 'email', 'password1', 'password2']

class Fuelxpress\_UserUpdateForm(forms.Modelform):

FE\_email = forms.EmailField()#email field same here

class meta:

FE\_model = User

FE\_fields = ['username', 'email']

class Fuelxpress\_ProfileUpdateForm(forms.ModelForm):

class Meta:

FE\_model = Fuelxpress\_Profile

FE\_fields = ['image']

**Users->views.py:**

* Using the views.py file under Users, **register(FE\_request)**: This part handles user registration.
* If someone is trying to register, it checks if the information they provided is correct. If the provided information is correct, it saves their registration. It then tells them that their account has been created successfully. Finally, it takes them to the login page.
* **profile(FE\_request)**: This part handles updating a user's profile.
* If someone is trying to update their profile, it checks if the information they provided is valid. If the provided information is valid, it updates their profile. It then tells them that their account has been updated successfully. Finally, it takes them back to their profile.
* The code makes sure that only users who are logged in can access their profile page.
* In the background, it uses forms to collect and validate user data, ensuring everything is in order.

def register(FE\_request):

# instantiate form with request data

if FE\_request.method == 'POST':

# if info is correct, save the form and fetch username

form = Fuelxpress\_UserRegisterForm(FE\_request.POST)

if form.is\_valid():

form.save()

FE\_username = form.cleaned\_data.get('username')

#Display the success message

messages.success(FE\_request, f'Your account has been created with Fuelxpress!')

return redirect('login')

#instantiate the blank form

else:

form = Fuelxpress\_UserRegisterForm()

return render(FE\_request, 'users/register.html', {'form': form})

#view for profile login, instantiate user and profile updated forms by POST

@login\_required

def profile(FE\_request):

if FE\_request.method == 'POST':

FE\_uform = Fuelxpress\_UserUpdateForm(FE\_request.POST, instance=FE\_request.user)

FE\_pform = Fuelxpress\_ProfileUpdateForm(FE\_request.POST,

FE\_request.FILES,

instance=FE\_request.user.profile)

if FE\_uform.is\_valid() and FE\_pform.is\_valid():

FE\_uform.save()

FE\_pform.save()

messages.success(FE\_request, f'Your account has been updated!')

return redirect('profile')

else:

FE\_uform = Fuelxpress\_UserUpdateForm(instance=FE\_request.user)

FE\_pform = Fuelxpress\_ProfileUpdateForm(instance=FE\_request.user.profile)

#Creating the context and render template

context = {

'u\_form': FE\_uform,

'p\_form': FE\_uform

}

return render(FE\_request, 'users/profile.html', context)

**Users->models.py:**  
**Fuelxpress\_Profile**, associated with the built-in User model. It includes a field for the user's profile image and overrides the **save** method to automatically resize and save profile images to a specified directory. The **\_\_str\_\_** method provides a string representation of the user's profile using their username. This model allows users to have profile pictures and ensures that the images are appropriately resized and stored.

from django.db import models

from django.contrib.auth.models import User

from PIL import Image

#It is o create profile model

class Fuelxpress\_Profile(models.Model):

FE\_user = models.OneToOneField(User, on\_delete=models.CASCADE)

FE\_image = models.ImageField(default='default.jpg', upload\_to='profile\_pics')

#To represent the string username

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

return f'{self.user.username} Profile'

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

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

FE\_img = Image.open(self.image.path)

#To check the size of image and resize it

if FE\_img.height > 300 or FE\_img.width > 300:

output\_size = (300, 300)

FE\_img.thumbnail(output\_size)

FE\_img.save(self.image.path)

**Users->signals.py**

Signals.py code having two signal handler methods:

create\_profile(sender, instance, created, \*\*kwargs): This method is called when a new user account is created. It checks if the user account was indeed created (created == True) and then creates a corresponding user profile for that user.

save\_profile(sender, instance, \*\*kwargs): This method is invoked when an existing user account is saved, such as when the user updates their information. It ensures that any modifications made to the related user profile are saved as well.

from django. import receiver

from .models import Profile

#it callbacks the receiver function for creating profile on new user

@Receiver(post\_save, sender=User\_name)

def create\_userprofile(sender, instance, created, \*\*kwargs):

if created:

Profile.objects.create(user=instance)

#to save the profile on saving user

@receiver(post\_save, sender=Username)

def save\_userprofile(sender, instance, \*\*kwargs):

#saving the profile linked to the instance

instance.profile.save()

**users->tests.py**

This code is a series of test cases for a Django application. It tests several components of the application, including views, forms, and models. Let's break down each test case:

1. **test\_registration\_view**: This method tests the registration view by sending a GET request to the 'register' URL and checking if the response status code is 200, indicating a successful page load.
2. **test\_profile\_view**: This function checks the profile view by first logging in a test user, then sending a GET request to the 'profile' URL. It verifies that the return status code is 302 (a redirect), and it guarantees that the response redirects to the 'login' page with a 'next' parameter set to the 'profile' URL.
3. **test\_user\_register\_form\_valid**: This method tests the validation of the user registration form by constructing a form with test data and claiming that the form is valid.
4. **test\_profile\_model**: This method verifies the profile model by establishing a test user and retrieving the associated profile. It then verifies that the string representation of the profile fits the required format, generally the username followed by "Profile."

from django.test import TestCase

from django.contrib.auth.models import User

from django.urls import reverse

from .models import Fuelxpress\_Profile

from .forms import Fuelxpress\_UserRegisterForm

class UsersAppTests(TestCase):

def test\_registration\_view(self):

FE\_response = self.client.get(reverse('register'))

self.assertEqual(FE\_response.status\_code, 200)

def test\_profile\_view(self):

self.client.login(username='testuser', password='testpwd')

FE\_response = self.client.get(reverse('profile'))

self.assertEqual(FE\_response.status\_code, 302)

self.assertRedirects(FE\_response, reverse('login') + '?next=' + reverse('profile'))

def test\_user\_register\_form\_valid(self):

form\_data = {

'username': 'testuser2',

'email': 'testuser2@example.com',

'password1': 'testpwd',

'password2': 'testpwd',

}

form = Fuelxpress\_UserRegisterForm(data=form\_data)

self.assertTrue(form.is\_valid())

def test\_profile\_model(self):

user = User.objects.create\_user(

username='testuser3',

email='testuser3@example.com',

password='testpassword'

)

profile = Fuelxpress\_Profile.objects.get(user=user)

self.assertEqual(str(profile), 'testuser3 Profile')

**users->urls.py**

This code having the URLs for register,profile,login,logout,password reset,password reset done,password reset confirm,password confirm view,password reset complete,password reset complete view.

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

from django.contrib.auth import views as auth\_views

from . import views

FE\_urlpatterns = [

Path\_FE('register/', views.register, name='register'),

Path\_FE('profile/', views.profile, name='profile'),

Path\_FE('login/', auth\_views.LoginView.as\_view(template\_name='users/login.html'), name='login'),

Path\_FE('logout/', auth\_views.LogoutView.as\_view(template\_name='users/logout.html'), name='logout'),

Path\_FE('password-reset/', auth\_views.PasswordResetView.as\_view(template\_name='users/password\_reset.html'),name='password\_reset'),

Path\_FE('password-reset/done/', auth\_views.PasswordResetDoneView.as\_view(template\_name='users/password\_reset\_done.html'),name='password\_reset\_done'),

Path\_FE('password-reset-confirm/<uidb64>/<token>/',auth\_views.PasswordResetConfirmView.as\_view(template\_name='users/password\_reset\_confirm.html' ),name='password\_reset\_confirm'),

Path\_FE('password-reset-complete/',auth\_views.PasswordResetCompleteView.as\_view(template\_name='users/password\_reset\_complete.html'),name='password\_reset\_complete'),

]

if settings.DEBUG:

FE\_urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Users->apps.py**

This part of code contains Django application configuration for users with userconfig class and ready method.

from django.apps import AppConfig

class UsersConfig(AppConfig):

name = 'users'

def ready(self):

import users.signals

**Users->admin.py**

In this code, two main classes are involved:

1. **admin.site**:

This is an instance of the Django admin site, which is used to manage and interact with the application's data through the admin interface. It includes many methods and functionalities for registering and handling models in the admin interface.

1. **Profile**:

This is the model class representing user profiles. The Profile model is defined in the application's models.py and is registered with the Django admin site using admin.site.register(Profile). This registration allows administrators to view and edit user profiles using the admin interface.

from django.contrib import admin #Import the admin module from django.contrib

from .models import Profile #Import the Profile model from the .models moduleadmin.site.register(Profile) #Register the Profile model with the admin site

**Dashboard->admin.py**

The code imports the admin module from the Django framework. The admin module gives administrative features for Django projects, allowing management and customization of the backend admin site.

**Code**:

from django.contrib import admin

# Registe the models.

#This module contains managing and administering the Django.

**Dashboard->apps.py**

Code imports the AppConfig class from the django.apps module. AppConfig is used to configure Django applications.New class called DashboardConfig is declared which inherits from AppConfig. This will be the configuration for an app named 'dashboard'.

The default\_auto\_field attribute is set to 'django.db.models.BigAutoField'. This specifies that future models created in this app will have a 64-bit auto-incrementing primary key by default.

The name attribute of the DashboardConfig class is set to 'dashboard'. This tells Django the name of the app that this configuration applies to.

Overall, this configures a new Django application called 'dashboard'. The app will use big auto-incrementing primary keys for models and the AppConfig subclass maintains the app-specific settings.

**Code**:

from django.apps import AppConfig #In this line we import Appconfig class from django.apps module.

class DashboardConfig(AppConfig):#we declare name of the class as a DashboardConfig that inherits from the Appconfig

default\_FE\_field = 'django.db.models.BigAutoField' #This line sets the default\_auto\_field attribute of the DashboardConfig class.

name = 'dashboard' #Name attribute of dashboardconfig class to dashboard.

**Dashboard->models.py**

The code imports the models module from the django.db package. This module offers the main tools for defining, generating, and changing model classes which represent database tables and fields.

**Code:**

from django.db import models

#this model contains classes and tools with database models.

# Create your models here.

**Dashboard->tests.py**

The code imports the TestCase class from the Django test framework and the reverse function from the Django URLs module.

Three test methods are defined:

* test\_index\_view: Sends a GET request to the index\_page view and asserts the response status code is 200 and uses the correct template.
* test\_home\_view: Sends a GET request to the home\_page view and asserts the response status code is 200 and uses the correct template.
* test\_about\_view: Sends a GET request to the about\_page view and asserts the response status code is 200 and uses the correct template.

**Code**:

from django.test import TestCase #we need to import TestCase class from Django test Framework.

from django.urls import reverse #we need to import the reverse function from the Django URL's Module.

class DashboardViewsTests(TestCase):#we create a class named as DashboardViewsTests which is inherits from the TestCase.

def test\_index\_view(self): #define a method to test the client to send a GET request.

response = self.client.get(reverse('index\_page'))

self.assertEqual(response.status\_view\_code, 200)

self.assertTemplateUsed(response, 'dashboard/index.html')

def test\_home\_view(self):# test client to send a GET request and http response code.

response = self.client.get(reverse('home\_page'))

self.assertEqual(response.status\_view\_code, 200)

self.assertTemplateUsed(response, 'dashboard/home.html')

def test\_about\_view(self):#Test client to send GET request and HTTP response.

response = self.client.get(reverse('about\_page'))

self.assertEqual(response.status\_view\_code, 200)

self.assertTemplateUsed(response, 'dashboard/about.html')

**Dashborad->views.py**

code imports the render function and get\_object\_or\_404 shortcut from Django shortcuts module. The User model is also imported from Django auth.

Three view functions are defined:

• index: Renders the 'dashboard/index.html' template in an HTTP response for the index page.

• home: Renders the 'dashboard/home.html' template in an HTTP response for the home page.

• about: Renders the 'dashboard/about.html' template in an HTTP response for the about page.

**Code**:  
from django.shortcuts import render, get\_object\_or\_404

from django.contrib.auth.models import User

def index(request):#Defining a function and rendering it using http response for index page.

return render(request, 'dashboard/index.html')

def home(request):##Defining a function and rendering it using http response for home page.

return render(request, 'dashboard/home.html')

def about(request):#Defining a function and rendering it using http response for about page.

return render(request, 'dashboard/about.html')