**Fuelxpressorder->admin**

Import Statements:

The code imports the admin module from django.contrib.

It also imports the FuelXpressOrder model from the same directory (.).

Model Registration:

The FuelXpressOrder model is registered with the Django admin site using admin.site.register(FuelXpressOrder).

This registration allows the model to be managed and viewed through the Django admin interface.

This code registers the FuelXpressOrder model with the Django admin site, enabling administrators to perform CRUD (Create, Read, Update, Delete) operations on instances of this model through the Django admin interface.

from django.contrib import admin#here i import required libraries

from .models import FuelXpressOrder

# you can register your models here.

admin.site.register(FuelXpressOrder)

**Fuelxpressorder->apps**

Import Statement:

The code imports the AppConfig class from django.apps.

Class Definition:

A new class named FuelxpressOrdersConfig is defined, which is expected to configure the Django application.

Class Attributes:

default\_auto\_field: This attribute is set to 'django.db.models.BigAutoField'. It specifies the default auto-incrementing primary key field for models in this application. In this case, it's set to the BigAutoField.

name: This attribute is set to 'fuelxpress\_orders'. It represents the name of the Django application.

This code defines a Django application configuration class (FuelxpressOrdersConfig), specifying certain configuration options for the application, such as the default auto-incrementing primary key field and the application name. This configuration class is likely used in the application's apps.py file to provide additional settings or metadata for the Django application.

from django.apps import AppConfig #importing the required configurations from django

class FuelxpressOrdersConfig(AppConfig):#creating a class name Fuelxpressordersconfig

default\_auto\_field = 'django.db.models.BigAutoField'

name = 'fuelxpress\_orders'

**Fuelxpressorder->forms**

Form Class Definition:

The class FuelXpressOrderForm is defined, inheriting from forms.ModelForm. This indicates that the form is a model form, closely tied to the FuelXpressOrder model.

Meta Class:

Inside the form class, there is a nested Meta class. This class is used to provide metadata for the form.

model: Specifies the model associated with this form (FuelXpressOrder).

fields: Lists the fields from the model that should be included in the form (['FX\_fuel\_type', 'FX\_gas\_station', 'FX\_driver', 'FX\_quantity', 'FX\_delivery\_location']).

Constructor (\_\_init\_\_) Method:

You can customize form field attributes by this overridden method.

The constructor calls the parent class constructor (super(FuelXpressOrderForm, self).\_\_init\_\_(\*args, \*\*kwargs)).

Then, it updates the attributes of specific fields ('FX\_fuel\_type', 'FX\_quantity', 'FX\_delivery\_location') by adding a CSS class 'form-control'. This is commonly used for styling in HTML forms.

This code defines a form class FuelXpressOrderForm for the FuelXpressOrder model, specifying which fields should be included in the form and customizing the attributes of certain fields for styling purposes.

from django import forms

from .models import FuelXpressOrder

class FuelXpressOrderForm(forms.ModelForm):#here i declare the class name as FuelXpressOrderForm.

class Meta:

#specifying the model that is associated with this form

model = FuelXpressOrder

fields = ['FX\_fuel\_type','FX\_gas\_station', 'FX\_driver', 'FX\_quantity', 'FX\_delivery\_location'] # Specify the fields required in the form

def \_\_init\_\_(self, \*args, \*\*kwargs):

#Calling the parent class constructor and updating the attributes of Fuelxpress

super(FuelXpressOrderForm, self).\_\_init\_\_(\*args, \*\*kwargs)

self.fields['FX\_fuel\_type'].widget.attrs.update({'class': 'form-control'})

self.fields['FX\_quantity'].widget.attrs.update({'class': 'form-control'})

self.fields['FX\_delivery\_location'].widget.attrs.update({'class': 'form-control'})

**Fuelxpressorder->models**

Import Statements:

The code imports necessary modules from Django, including models, timezone, User model for authentication, and modules from other apps (FuelXpressGasStation and FuelXpressDriver).

Model Definition:

The FuelXpressOrder class is defined, inheriting from models.Model.

Several fields are defined within the class to represent various attributes of an order, including a foreign key relationship to the User model for the user placing the order.

Fields include FX\_gas\_station, FX\_driver, FX\_fuel\_type, FX\_quantity, FX\_delivery\_location, FX\_order\_status, and FX\_order\_date.

\_\_str\_\_ Method:

The \_\_str\_\_ method is defined to provide a human-readable string representation of the model instance. It returns a string indicating the username associated with the order.

This code defines a Django model (FuelXpressOrder) representing orders for fuel delivery. It includes various fields, such as user, gas station, driver, fuel type, quantity, delivery location, order status, and order date. The \_\_str\_\_ method provides a human-readable string representation for instances of this model.

#importing the required modules from django

from django.db import models

from django.utils import timezone

from django.contrib.auth.models import User

from fuelxpress\_gasstations.models import FuelXpressGasStation

from fuelxpress\_drivers.models import FuelXpressDriver

from django.urls import reverse

#Defining the FuelXpressorder model

class FuelXpressOrder(models.Model):

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

#Defining the foreign key relationship

FX\_gas\_station = models.ForeignKey(FuelXpressGasStation, on\_delete=models.CASCADE)

FX\_driver = models.ForeignKey(FuelXpressDriver, on\_delete=models.CASCADE)

#Definig the character field

FX\_fuel\_type = models.CharField(max\_length=100)

FX\_quantity = models.DecimalField(max\_digits=10, decimal\_places=2)

FX\_delivery\_location = models.TextField()

FX\_order\_status = models.CharField(max\_length=50, default="Pending")

#Defining the date-time field to store

FX\_order\_date = models.DateTimeField(default=timezone.now)

#Definig the human-redable string representation

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

return f"Order for {self.FX\_user.username}"

**Fuelxpressorder->tests**

Test Case Class:

The OrderTestCase class inherits from django.test.TestCase, indicating that it is a Django test case.

setUp Method:

The setUp method is used to set up the initial conditions for the test case.

It creates a sample user, gas station, driver, and an order instance for testing.

Test Functions:

test\_order\_user, test\_order\_gas\_station, test\_order\_driver, and test\_order\_fuel\_type are individual test functions.

Each function tests a specific aspect of the FuelXpressOrder model instance created in the setUp method.

These tests use assertions to check if the values stored in the model instance match the expected values.

This test case class is designed to test different aspects of the FuelXpressOrder model, including the association with a user, gas station, driver, and the correct storage of fuel type.

from django.test import TestCase # importing library for test case.

from .models import FuelXpressOrder

from django.contrib.auth.models import User

from fuelxpress\_gasstations.models import FuelXpressGasStation

from fuelxpress\_drivers.models import FuelXpressDriver

from django.utils import timezone

class OrderTestCase(TestCase):

def setUp(self):

# Create a sample user

self.user = User.objects.create\_user(username="testuser", password="testpassword")

# Create a sample GasStation with inventory

self.gas\_station = FuelXpressGasStation.objects.create(

FX\_name="Test Gas Station",

FX\_phone\_number="123-456-7890",

FX\_email="test@example.com",

FX\_location="123 Test Street",

FX\_inventory=1000.00,

)

# Create a sample Driver

self.driver = FuelXpressDriver.objects.create(FX\_name="Test Driver", FX\_license\_number="12345")

# Create a sample Order instance for testing

self.order = FuelXpressOrder(

FX\_user=self.user,

FX\_gas\_station=self.gas\_station,

FX\_driver=self.driver,

FX\_fuel\_type="Regular",

FX\_quantity=10.5,

FX\_delivery\_location="123 Test Street",

FX\_order\_status="Pending",

FX\_order\_date=timezone.now(),

)

self.order.save()

# here we define different types of functions for user,gas station ,order driver and order fuel type.

def test\_order\_user(self):

order = FuelXpressOrder.objects.get(FX\_fuel\_type="Regular")

self.assertEqual(order.FX\_user, self.user)

def test\_order\_gas\_station(self):

order = FuelXpressOrder.objects.get(FX\_fuel\_type="Regular")

self.assertEqual(order.FX\_gas\_station, self.gas\_station)

def test\_order\_driver(self):

order = FuelXpressOrder.objects.get(FX\_fuel\_type="Regular")

self.assertEqual(order.FX\_driver, self.driver)

def test\_order\_fuel\_type(self):

order = FuelXpressOrder.objects.get(FX\_fuel\_type="Regular")

self.assertEqual(order.FX\_fuel\_type, "Regular")

**Fuelxpressorder->urls**

Import Statements:

The code imports the path function from django.urls.

It also imports the views module from the current directory (., which typically refers to the Django app).

URL Patterns:

urlpatterns is a list of URL patterns for FuelXpress orders.

Path Patterns:

path('create/', views.fuelxpress\_create\_order, name='fuelxpress\_create\_order'): This pattern maps to a view function fuelxpress\_create\_order for creating a new order. The name 'fuelxpress\_create\_order' can be used in Django templates or other parts of the code to refer to this URL.

path('<int:order\_id>/', views.fuelxpress\_order\_detail, name='fuelxpress\_order\_detail'): This pattern includes a dynamic segment <int:order\_id> to capture an integer value as order\_id and maps to a view function fuelxpress\_order\_detail for viewing details of a specific order. The name is set to 'fuelxpress\_order\_detail'.

path('list/', views.fuelxpress\_order\_list, name='fuelxpress\_order\_list'): This pattern maps to a view function fuelxpress\_order\_list for listing all orders. The name is set to 'fuelxpress\_order\_list'.

These URL patterns are meant to be included in the main urls.py file of the Django project or in the urls.py file of the app. They define the routing for FuelXpress orders, specifying which view function to call for different actions like creating, viewing details, and listing orders.

from django.urls import path

from . import views

# urlpatterns for FuelXpress orders

urlpatterns = [

#urlpatterns for creating a new order

path('create/', views.fuelxpress\_create\_order, name='fuelxpress\_create\_order'),

# urlpatterns for viewing details of a specific order

path('<int:order\_id>/', views.fuelxpress\_order\_detail, name='fuelxpress\_order\_detail'),

# urlpatterns for listing all orders

path('list/', views.fuelxpress\_order\_list, name='fuelxpress\_order\_list'),

]

**Fuelxpressorder->views**

#Importing the necessary modules

from django.shortcuts import render, redirect

from .forms import FuelXpressOrderForm

from django.shortcuts import get\_object\_or\_404

from .models import FuelXpressOrder

from django.contrib.auth.decorators import login\_required

@login\_required

def fuelxpress\_create\_order(request):#here we create function named as fuelxpress\_create\_order.

if request.method == 'POST':

form = FuelXpressOrderForm(request.POST)

if form.is\_valid():

fuelxpress\_order = form.save(commit=False)

fuelxpress\_order.FX\_user = request.user # Set the user associated with the order

fuelxpress\_order.save()

return redirect('fuelxpress\_order\_detail', order\_id=fuelxpress\_order.pk)

else:

form = FuelXpressOrderForm()

return render(request, 'fuelxpress\_orders/fuelxpress\_order\_create.html', {'form': form})

@login\_required

def fuelxpress\_order\_detail(request, order\_id):#here we create function named as fuelxpress\_order\_detail.

order = get\_object\_or\_404(FuelXpressOrder, pk=order\_id)

return render(request, 'fuelxpress\_orders/fuelxpress\_order\_detail.html', {'order': order})

@login\_required

def fuelxpress\_order\_list(request):#here we create function named as fuelxpress\_order\_list.

orders = FuelXpressOrder.objects.filter(FX\_user=request.user) # Use the correct field name here

return render(request, 'fuelxpress\_orders/fuelxpress\_order\_list.html', {'orders': orders})

Import Statements:

The code imports necessary modules, including functions from django.shortcuts, the FuelXpressOrderForm form, the get\_object\_or\_404 function, the FuelXpressOrder model, and the login\_required decorator.

Views:

fuelxpress\_create\_order: This view is responsible for creating a new order. It handles both GET and POST requests. If the form is submitted with valid data, a new order is created and associated with the logged-in user. The user is then redirected to the order detail page.

fuelxpress\_order\_detail: This view displays details of a specific order. It uses the get\_object\_or\_404 function to retrieve the order with the given order\_id or return a 404 page if not found.

fuelxpress\_order\_list: This view lists all orders associated with the logged-in user. It retrieves orders using FuelXpressOrder.objects.filter(FX\_user=request.user).

login\_required Decorator:

All three views (fuelxpress\_create\_order, fuelxpress\_order\_detail, fuelxpress\_order\_list) are decorated with @login\_required, ensuring that only authenticated users can access these views.

Template Rendering:

The views render HTML templates located in the 'fuelxpress\_orders' directory, passing necessary data to the templates.

In summary, these views handle the creation, detail display, and listing of FuelXpress orders. The login\_required decorator ensures that only authenticated users can access these views.