Here’s the equivalent Django implementation for both the `store` and `handlePaymentConfirmation` functions. I'll use Django's `views.py` and `models.py` to replicate the functionality.

---

### `models.py`

```python

from django.db import models

import hashlib

import time

import random

class PayNowPayment(models.Model):

regnum = models.CharField(max\_length=50)

amount = models.FloatField()

reference = models.CharField(max\_length=64, unique=True)

status = models.CharField(max\_length=50, default="created")

browserurl = models.URLField(blank=True, null=True)

pollurl = models.URLField(blank=True, null=True)

hash = models.CharField(max\_length=64, blank=True, null=True)

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

return f"Payment {self.reference} - {self.status}"

class StudentPackage(models.Model):

reference = models.CharField(max\_length=64)

paid = models.IntegerField(default=0)

regnum = models.CharField(max\_length=50)

packageid = models.CharField(max\_length=50)

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

return f"Package {self.packageid} for {self.regnum}"

```

---

### `views.py`

```python

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

from django.http import JsonResponse, HttpResponseServerError

from django.core.exceptions import ValidationError

from django.db import transaction

from .models import PayNowPayment, StudentPackage

import hashlib

import time

import random

# Mock PayNow class (replace with actual implementation)

class PayNow:

def \_\_init\_\_(self, integration\_id, integration\_key, return\_url, result\_url):

self.integration\_id = integration\_id

self.integration\_key = integration\_key

self.return\_url = return\_url

self.result\_url = result\_url

def create\_payment(self, narration, email):

# Mock implementation

return Payment(narration, email)

def send(self, payment):

# Mock implementation

return PaymentResponse()

def poll\_transaction(self, poll\_url):

# Mock implementation

return PollResponse()

class Payment:

def \_\_init\_\_(self, narration, email):

self.narration = narration

self.email = email

def add(self, narration, amount):

# Mock implementation

pass

class PaymentResponse:

def success(self):

# Mock implementation

return True

def redirect\_url(self):

# Mock implementation

return "https://example.com/redirect"

def poll\_url(self):

# Mock implementation

return "https://example.com/poll"

def data(self):

# Mock implementation

return {"hash": "mock\_hash"}

class PollResponse:

def status(self):

# Mock implementation

return "Paid"

def paynow\_reference(self):

# Mock implementation

return "mock\_reference"

# Django settings for PayNow integration

PAYNOW\_INTEGRATION\_ID = "your\_integration\_id"

PAYNOW\_INTEGRATION\_KEY = "your\_integration\_key"

APP\_URL2 = "https://example.com"

def store(request):

if request.method == "POST":

regnum = request.POST.get("regnum")

price = request.POST.get("price")

packageid = request.POST.get("packageid")

package = request.POST.get("package")

myname = request.POST.get("myname")

# Validation

if not regnum or not price or not price.isdigit() or float(price) < 1:

return JsonResponse({"error": "Invalid input"}, status=400)

# Generate unique reference

unique\_string = hashlib.sha256(f"{time.time()}{random.randint(0, 1000000)}".encode()).hexdigest()

# Prepare description and narration

desc = f"{myname} ({regnum})"

narration = f"{package} package payment for {desc}"

# Save payment to database

payment\_saved = PayNowPayment(

regnum=regnum,

amount=float(price),

reference=unique\_string,

status="created"

)

payment\_saved.save()

# Initialize PayNow

paynow = PayNow(

PAYNOW\_INTEGRATION\_ID,

PAYNOW\_INTEGRATION\_KEY,

f"{APP\_URL2}/payconfirmation/{payment\_saved.id}",

f"{APP\_URL2}/payment\_return"

)

# Create and send payment

payment = paynow.create\_payment(narration, 'netecsystems@gmail.com')

payment.add(narration, float(price))

try:

response = paynow.send(payment)

except Exception as e:

return JsonResponse({"error": str(e)}, status=500)

if response.success():

# Update payment with response details

payment\_saved.browserurl = response.redirect\_url()

payment\_saved.pollurl = response.poll\_url()

payment\_saved.hash = response.data()["hash"]

payment\_saved.save()

# Update or create student package

StudentPackage.objects.update\_or\_create(

reference=unique\_string,

defaults={"paid": 0, "regnum": regnum, "packageid": packageid}

)

return redirect(response.redirect\_url())

return JsonResponse({"error": "Payment processing failed"}, status=400)

return JsonResponse({"error": "Invalid request method"}, status=405)

def handle\_payment\_confirmation(request, id):

payment = get\_object\_or\_404(PayNowPayment, id=id)

reference = ""

status = ""

try:

# Poll transaction status

poll = PayNow(PAYNOW\_INTEGRATION\_ID, PAYNOW\_INTEGRATION\_KEY, "", "").poll\_transaction(payment.pollurl)

payment.status = status = poll.status()

reference = payment.reference

payment.save()

except Exception as e:

return HttpResponseServerError(str(e))

# Check if payment is successful

if status == "Paid":

# Update payment status

payment.status = "Paid"

payment.save()

# Update student package status

StudentPackage.objects.filter(reference=reference).update(paid=1)

return JsonResponse({"message": "Payment confirmed and saved successfully"}, status=200)

else:

return JsonResponse({"error": "Payment was not successful"}, status=400)

```

---

### `urls.py`

Add the following routes to your `urls.py`:

```python

from django.urls import path

from . import views

urlpatterns = [

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

path('handle\_payment\_confirmation/<int:id>/', views.handle\_payment\_confirmation, name='handle\_payment\_confirmation'),

]

```

---

### Key Points:

1. \*\*Models\*\*:

- `PayNowPayment` and `StudentPackage` represent the database tables.

- Fields like `reference`, `status`, and `pollurl` are used to store payment details.

2. \*\*Views\*\*:

- `store`: Handles the payment initiation and saves the payment details.

- `handle\_payment\_confirmation`: Polls the payment status and updates the database accordingly.

3. \*\*Mock PayNow Class\*\*:

- Replace the `PayNow`, `Payment`, `PaymentResponse`, and `PollResponse` classes with the actual implementation of your payment gateway.

4. \*\*Validation\*\*:

- Input validation is performed to ensure required fields are present and valid.

5. \*\*Database Updates\*\*:

- `update\_or\_create` is used to update or insert records in the `StudentPackage` table.

6. \*\*Error Handling\*\*:

- Exceptions are caught and handled gracefully, returning appropriate HTTP status codes and error messages.

---