```
from django.http import JsonResponse, HttpResponse
from django.shortcuts import render, get_object_or_404, redirect
#from rest_framework import viewsets
from django.views.decorators.csrf import csrf_exempt
#from django_filters.rest_framework import DjangoFilterBackend
#from .serializers import RecepiSerializer
from .models import Recepi, Compound, RawMat, RawGrp,
MixProd
from .forms import RecepiForm, MixProdForm
from .utils import render_to_pdf
from django.contrib.auth.decorators import login_required
from django.utils.dateparse import parse_date

from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.pagesizes import A4, letter
from reportlab.platypus import Paragraph
from reportlab.pdfgen import canvas
from reportlab.lib import colors
from reportlab.lib.units import cm
import base64
import json

import pandas as pd
from io import BytesIO
from reportlab.graphics import renderPM
from barcode import Code128

import os
import re
import barcode
from barcode.writer import ImageWriter
from django.conf import settings

import aiml

import fitz  # PyMuPDF for extracting links from PDF
from django.core.files.storage import default_storage
from django.core.files.base import ContentFile
from django.http import JsonResponse
```

```
import requests
import mimetypes
import magic  # pip install python-magic-bin
              # Python-magic for detecting file types
from urllib.parse import urlparse



#https://github.com/Sunil-Chakraborty/pyweb

# List View

def recepi_list(request):
    recepis = Recepi.objects.select_related('comp_cd', 'rm_cd',
'rm_cd__grp_cd').all()
    comp_cds = Compound.objects.all()

    rm_cds = RawMat.objects.all().order_by('rm_cd')

    pdf_directory = os.path.join(settings.MEDIA_ROOT, 'resources',
'pdf')
    pdf_files = []

    try:
        # Generate full URLs for each file to use in the template
        pdf_files = [
            {
                'name': f,
                'url': os.path.join(settings.MEDIA_URL, 'resources', 'pdf',
f)
            }
            #for f in os.listdir(pdf_directory) if f.endswith('.pdf')
            for f in os.listdir(pdf_directory)
        ]
    except FileNotFoundError:
        pass

    return render(request, 'product/recepi_list.html', {
```

```
    'recepis' : recepis,
    'comp_cds' : comp_cds,
    'rm_cds'   : rm_cds,
    'pdf_files': pdf_files,
    })

# Add/Edit View
def recepi_add(request):


    if request.method == 'POST':
        # Get the header fields

        card_no = request.POST.get('card_no')
        doc_dt = request.POST.get('doc_dt')
        comp_cd = request.POST.get('comp_cd')

        # Get the dynamically added raw materials and quantities
        rm_cds = request.POST.getlist('rm_cd[]')
        qtys = request.POST.getlist('qty[]')

        # Process each row (raw material and quantity)
        for rm_cd, qty in zip(rm_cds, qtys):
            Recepi.objects.create(
                card_no=card_no,
                doc_dt=doc_dt,
                comp_cd_id=comp_cd,  # Assuming ForeignKey to
Compound model
                rm_cd_id=rm_cd,  # Assuming ForeignKey to RawMat
model
                qty=qty
            )

        return redirect('product:recepi_list')

    else:
        comp_cds = Compound.objects.all()
        rm_cds = RawMat.objects.all()
```

```
    return render(request, 'product/recepi_list.html', {
        'comp_cds': comp_cds,
        'rm_cds': rm_cds,
    })

@csrf_exempt
def recepi_edit(request, recepi_id):
    if request.method == 'POST':
        comp_id = request.POST.get('compid')
        rm_id = request.POST.get('rmcdid')
        qty = request.POST.get('qty')
        doc_date = request.POST.get('created_date')  # Ensure this is
correctly retrieved


        # Parse the date string to a proper date object if it's a valid
string
        """
        if doc_date:
            if isinstance(doc_date, str):
                doc_date = parse_date(doc_date)
            else:
                doc_date = str(doc_date)
        """

        # Get the Recepi object and update its fields
        recepi = get_object_or_404(Recepi, id=recepi_id)
        recepi.comp_cd_id = comp_id
        recepi.rm_cd_id = rm_id
        recepi.qty = qty
        #recepi.created_date = doc_date  # Assign the date here
        recepi.save()

        #return redirect('product:recepi_list')
        return redirect(request.path)


    return redirect('product:recepi_list')
```

```
# Delete View
"""
def recepi_delete(request, id):
    recepi = get_object_or_404(Recepi, id=id)
    if request.method == 'POST':
        recepi.delete()
        return redirect('product:recepi_list')
    return render(request, 'product/recepi_confirm_delete.html',
{'recepi': recepi})
"""
def recepi_delete(request, id):
    recepi = get_object_or_404(Recepi, id=id)
    recepi.delete()
    return redirect('product:recepi_list')

@csrf_exempt
def generate_pdf(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body).get('data', [])
            #print(data)
        except json.JSONDecodeError:
            data = []

        if not data:
            return HttpResponse('No data provided', status=400)

        response = HttpResponse(content_type='application/pdf')
        response['Content-Disposition'] = 'attachment;
filename="Recepi_Report.pdf"'

        p = canvas.Canvas(response, pagesize=A4)
        width, height = A4

        # Title
        p.setFont("Helvetica-Bold", 16)
        p.drawString(2*cm, height - 2*cm, "Compound Recepi Report")
```

```
        # Display Compound Code
        p.setFont("Helvetica-Bold", 12)
        compound_code = data[0][1]  # Assuming the first item in data
is the Compound Code
        sap_mix = data[0][2]  # Assuming the first item in data is the
Compound Code
        p.drawString(2*cm, height - 3*cm, f"Comp. Code :
{compound_code}   SAP CD: {sap_mix}")

        # Adjust position for headers
        headers_y_position = height - 4*cm
        p.setFont("Helvetica-Bold", 10)
        p.drawString(2*cm, headers_y_position, "R.M Code")
        p.drawString(4*cm, headers_y_position, "SAP Code")
        p.drawString(7*cm, headers_y_position, "Description")
        p.drawString(12*cm, headers_y_position, "UOM")
        p.drawRightString(14*cm, headers_y_position, "Qty.")
        p.drawRightString(16*cm, headers_y_position, "Rs./Kg.")
        p.drawRightString(18*cm, headers_y_position, "Amount")

        # Draw line under headers
        p.line(2*cm, headers_y_position - 5, width - 2*cm,
headers_y_position - 5)

        # Set position for data rows
        y_position = headers_y_position - 1*cm
        p.setFont("Helvetica", 10)

        total_quantity = 0
        total_amount = 0

        # Sample StyleSheet for Paragraph
        styles = getSampleStyleSheet()
        normal_style = styles["Normal"]

        for row in data:
            if y_position < 2*cm:
                p.showPage()
                p.setFont("Helvetica", 10)
```

```
            y_position = height - 4*cm
            p.drawString(2*cm, y_position, "R.M Code")
            p.drawString(4*cm, y_position, "SAP Code")
            p.drawString(7*cm, y_position, "Description")
            p.drawString(12*cm, y_position, "UOM")
            p.drawRightString(14*cm, y_position, "Qty.")
            p.drawRightString(16*cm, y_position, "Rs./Kg.")
            p.drawRightString(18*cm, y_position, "Amount")
            p.line(2*cm, y_position - 5, width - 2*cm, y_position - 5)
            y_position -= 1*cm

        # Calculate Amount
        quantity = float(row[8])
        rs_per_kg = float(row[9])
        amount = quantity * rs_per_kg



        # Draw the data values
        p.drawString(2*cm, y_position, row[3])  # R.M Code
        p.drawString(4*cm, y_position, row[4])  # SAP Code

        # Wrap description with Paragraph
        description = Paragraph(row[5], normal_style)
        description.wrap(4*cm, 1*cm)  # Set width and height

        # Calculate the space needed to wrap the text
        text_width, text_height = description.wrap(4*cm, 1*cm)  #
4cm width, 1cm height constraint
        y_pos=y_position
        if text_height > 15:
            y_pos -= .5*cm
        else:
            y_pos=y_position

        description.drawOn(p, 7*cm, y_pos)

        p.drawString(12*cm, y_position, row[7])  # UOM
        p.drawRightString(14*cm, y_position, f"{quantity:.2f}")  #
```

```
Quantity
            p.drawRightString(16*cm, y_position, f"{rs_per_kg:.2f}")  #
Rs./Kg.
            p.drawRightString(18*cm, y_position, f"{amount:.2f}")  #
Amount

            # Accumulate totals
            total_quantity += quantity
            total_amount += amount

            y_position -= 1*cm

        # Draw line above totals
        p.line(2*cm, y_position, width - 2*cm, y_position)  # Draw line
from left to right

        # Move down for totals
        #y_position -= 0.5*cm

        # Draw totals
        p.setFont("Helvetica-Bold", 10)
        y_position -= 1*cm  # Adjust position for totals
        p.drawString(2*cm, y_position, "Totals:")
        p.drawRightString(14*cm, y_position, f"{total_quantity:.2f}")  #
Total Quantity
        p.drawRightString(16*cm, y_position, "")  # Leave Rs./Kg.
blank for total row
        p.drawRightString(18*cm, y_position, f"{total_amount:.2f}")  #
Total Amount

        p.showPage()
        p.save()

        return response
    else:
        return HttpResponse('Invalid request method', status=405)

@csrf_exempt
def generate_excel(request):
```

```
    try:
        if request.method == 'POST':
            # Parse data from the request
            data = json.loads(request.body).get('data', [])
            if not data:
                return HttpResponse('No data provided', status=400)

            # Define the correct column headers
            columns = ['R.M Code', 'Description', 'UOM', 'Quantity',
'Rate']

            # Create a Pandas DataFrame from the data
            df = pd.DataFrame(data, columns=columns)
            print(df)

            # Create an in-memory Excel file
            excel_file = BytesIO()

            # Use ExcelWriter to write the DataFrame to an Excel file
            with pd.ExcelWriter(excel_file, engine='xlsxwriter') as writer:
                df.to_excel(writer, index=False,
sheet_name='Recepi_Report')

            # Ensure the file pointer is at the beginning of the BytesIO
object
            excel_file.seek(0)

            # Return the Excel file as a response
            response = HttpResponse(excel_file.getvalue(),
content_type='application/vnd.openxmlformats-officedocument.spre
adsheetml.sheet')
            response['Content-Disposition'] = 'attachment;
filename="Recepi_Report.xlsx"'
            return response

        else:
            return HttpResponse('Invalid request method', status=405)

    except Exception as e:
```

```
        print(f"Error generating Excel: {e}")
        return HttpResponse(f"Error generating Excel: {e}",
status=500)


def generate_labels(request):
    # Fetch data from the database
    rawmat_data = RawMat.objects.all()  # Adjust the query as per
your needs

    # Pass the data to the template
    return render(request, 'product/labels_template.html',
{'rawmat_data': rawmat_data})

@csrf_exempt
def generate_html_report(request):
    if request.method == 'POST':
        try:
            data = json.loads(request.body).get('data', [])
        except json.JSONDecodeError:
            data = []

        if not data:
            return JsonResponse({'error': 'No data provided'},
status=400)

        # Initialize totals
        total_qty = 0.0
        total_amt = 0.0

        # Process the data
        processed_data = []
        for row in data:

            # Add data processing logic here
            quantity = float(row[8])
            rs_per_kg = float(row[9])
            amount = quantity * rs_per_kg
```

```
            # Add data processing logic here
            processed_data.append({
                'comp_code': row[1],
                'sap_code': row[2],
                'rm_code': row[3],
                'sap_rm_code': row[4],
                'description': row[5],
                'group_desc': row[6],
                'uom': row[7],
                'quantity': f"{float(row[8]):.2f}",
                'rs_per_kg': f"{float(row[9]):.2f}",
                'amount': f"{float(row[8]) * float(row[9]):.2f}"

            })

            total_qty += quantity
            total_amt += amount

        # Render the HTML template with the processed data
        return render(request, 'product/recepi_doc.html', {
            'data': processed_data,
            'compound_code': processed_data[0]['comp_code'] if
processed_data else '',
            'sap_code': processed_data[0]['sap_code'] if
processed_data else '',
            'total_qty': f"{total_qty:.2f}",
            'total_amt': f"{total_amt:.2f}",

        })

    elif request.method == 'GET':
        # Return the rendered HTML in response to a GET request
        print("GET")
        return render(request, 'product/recepi_doc.html')

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

#ZXing Decoder Online (https://zxing.org/w/decode.jspx)
```

```
def sanitize_input(input_string):
    # Replace invalid characters like µ (or any others you don't want)
    return re.sub(r'[^a-zA-Z0-9 -]', '', input_string)  # Keeps letters, numbers, spaces, and hyphens

def generate_barcodes_view(request):
    barcode_dir = os.path.join(settings.MEDIA_ROOT, 'barcodes')  # Directory for barcode images
    pdf_path = os.path.join(barcode_dir, "labels.pdf")  # Path for the generated PDF
    writer_options = {'write_text': False}  # Set writer options to hide the text

    # Ensure barcode directory exists
    if not os.path.exists(barcode_dir):
        os.makedirs(barcode_dir)

    # Set up the PDF canvas
    c = canvas.Canvas(pdf_path, pagesize=letter)
    width, height = letter
    y_position = height - 50

    # Iterate over all Raw Materials to generate barcodes and labels
    for raw_mat in RawMat.objects.all():
        barcode_content = f"{raw_mat.rm_cd} | {sanitize_input(raw_mat.rm_des)} | {raw_mat.uom} | {raw_mat.rate}"
        barcode_class = barcode.get_barcode_class('code128')
        barcode_obj = barcode_class(barcode_content, writer=ImageWriter())

        # Construct the file path without .png extension
        barcode_image_path = os.path.join(barcode_dir, raw_mat.rm_cd)

        # Save the barcode image
        saved_file = barcode_obj.save(barcode_image_path, options=writer_options)
        barcode_image_full_path = saved_file
```

```
        # Check if the barcode image file exists
        if os.path.exists(barcode_image_full_path):
            # Draw the Raw Material Description and Code on the PDF
            c.drawString(100, y_position, f"Desc.: {raw_mat.rm_des}
UOM:{raw_mat.uom}")
            c.drawString(100, y_position - 15, f"Code: {raw_mat.rm_cd}
Rate: {raw_mat.rate} ")


            # Draw the barcode image below the text
            c.drawImage(barcode_image_full_path, 100, y_position -
68, width=150, height=50)

        # Adjust position for the next label
        y_position -= 150

        if y_position < 100:
            c.showPage()  # Start a new page if out of space
            y_position = height - 50

    # Save the PDF
    c.save()

    # Return the PDF as a downloadable response
    with open(pdf_path, 'rb') as pdf_file:
        response = HttpResponse(pdf_file.read(),
content_type='application/pdf')
        response['Content-Disposition'] = f'inline; filename="labels.pdf"'
        return response

# Bar decoder : https://zxing.org/w/decode.jspx

@csrf_exempt
def generate_doc_report(request):

    if request.method == 'POST':
        try:
            data = json.loads(request.body).get('data', [])
```

```
        except json.JSONDecodeError:
            data = []

        if not data:
            return JsonResponse({'error': 'No data provided'},
status=400)

        # Initialize totals
        total_qty = 0.0
        total_amt = 0.0

        # Process the data
        processed_data = []
        for row in data:
            quantity = float(row[10])
            rs_per_kg = float(row[11])
            amount = quantity * rs_per_kg

            # Generate barcode on the fly for SAP code (row[2])
            barcode_buffer = BytesIO()
            # Set the options for higher resolution
            options = {
                'module_width': 0.2,  # Default is 0.2, you can reduce it for
a finer barcode
                'module_height': 5,  # Adjust height
                'font_size': 0,  # Hide the text
                'dpi': 300,  # Increase the DPI for better quality
                'write_text': False  # Don't write the text below the barcode
            }

            barcode_data = row[5] + row[6]  # Concatenating row[3] (RM
Code) and row[4] (SAP RM Code)

            #barcode_data = f"row[3]"

            # Create a barcode object and adjust size with options
            barcode_obj = Code128(barcode_data,
writer=ImageWriter())
```

```
            barcode_obj.write(barcode_buffer, options=options)

            # Convert the barcode image to base64 so it can be
embedded in HTML
            barcode_image_base64 =
base64.b64encode(barcode_buffer.getvalue()).decode('utf-8')
            barcode_image_base64 =
f"{barcode_image_base64}"

            # Add data processing logic here
            processed_data.append({
                'comp_code': row[3],
                'bar_code': barcode_image_base64,  # Replace SAP
code with barcode image
                'sap_code': row[4],
                'rm_code': row[5],
                'sap_rm_code': row[6],
                'description': row[7],
                'group_desc': row[8],
                'uom': row[9],
                'quantity': f"{float(row[10]):.3f}",
                'rs_per_kg': f"{float(row[11]):.2f}",
                'amount': f"{float(row[10]) * float(row[11]):.2f}"
            })

            total_qty += quantity
            total_amt += amount

        # Render the HTML template with the processed data
        #return render(request, 'product/recepi_doc_barcode.html', {
        return render(request, 'product/recepi_card.html', {
            'data': processed_data,
            #'compound_code': processed_data[1]['comp_code'] if
processed_data else '',
            #'bar_code': processed_data[0]['sap_code'] if
processed_data else '',
            'compound_code': row[3],
            'sap_code': row[4],
```

```
            'total_qty': f"{total_qty:.3f}",
            'total_amt': f"{total_amt:.2f}"
        })

    elif request.method == 'GET':
        # Return the rendered HTML in response to a GET request
        return render(request, 'product/recepi_card.html.html')
        #return render(request, 'product/recepi_doc_barcode.html')

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




# List all production orders
def mixprod_list(request):
    mixprods = MixProd.objects.all()

    # Fetch all card_no and comp_cd combinations
    recepi_queryset = Recepi.objects.all()

    # Use a set to store unique card_no values
    unique_recepi = []
    seen_card_nos = set()

    for recepi in recepi_queryset:
        if recepi.card_no not in seen_card_nos:
            unique_recepi.append(recepi)
            seen_card_nos.add(recepi.card_no)  # Add the card_no to
the set

    #recepi_queryset = Recepi.objects.values('card_no',
'comp_cd').distinct()

    context = {
        'mixprods': mixprods,
        'recepi_queryset': unique_recepi,  # Pass the querysets to the
```

```
template

    }
    return render(request, 'product/mixprod_list.html', context)

# Create a new production order
def mixprod_create(request):
    if request.method == 'POST':
        form = MixProdForm(request.POST)

        if form.is_valid():
            # Retrieve cleaned data from the form
            recepi_instance = form.cleaned_data['card_no']  # This is
already a Recepi instance
            prod_no = form.cleaned_data['prod_no']
            prod_dt = form.cleaned_data['prod_dt']
            qty = form.cleaned_data['qty']
            print(form.cleaned_data['card_no'])
            # Create MixProd with the Recepi instance
            MixProd.objects.create(
                card_no=recepi_instance,  # Use Recepi instance here
                prod_no=prod_no,
                prod_dt=prod_dt,
                qty=qty
            )

            return redirect('product:mixprod_list')

        else:
            # Debugging: see what caused the form to fail

            print("Form errors:", form.errors)

    else:
        form = MixProdForm()

    return render(request, 'product/mixprod_form.html', {'form': form})
```

```
# Edit an existing production order
def mixprod_edit(request, pk):
    mixprod = get_object_or_404(MixProd, pk=pk)
    if request.method == 'POST':
        form = MixProdForm(request.POST, instance=mixprod)
        if form.is_valid():
            form.save()
            return redirect('product:mixprod_list')
        else:
            form = MixProdForm(instance=mixprod)
    else:
        form = MixProdForm(instance=mixprod)
    return render(request, 'product/mixprod_list.html', {'form': form})

# Delete a production order
def mixprod_delete(request, pk):
    mixprod = get_object_or_404(MixProd, pk=pk)
    if request.method == 'POST':
        mixprod.delete()
        return redirect('product:mixprod_list')
    return render(request, 'product/mixprod_confirm_delete.html',
{'mixprod': mixprod})



def mixprod_view(request, pk):
    # Get the production order
    mixprod = get_object_or_404(MixProd, pk=pk)

    # Get all recepi items related to the card_no of the production
order
    recepi_items =
Recepi.objects.filter(card_no=mixprod.card_no.card_no)

    # Calculate the total quantity of raw materials (recepi qty)
    total_recepi_qty = sum(item.qty for item in recepi_items)

    # Calculate the BOM factor (Production Order qty / Total Recepi
qty)
```

```
    bom_factor = mixprod.qty / total_recepi_qty if total_recepi_qty > 0
else 0

    total_proportionate_qty = sum(bom_factor * item.qty for item in
recepi_items)  # Sum of proportionate quantities

    # Calculate proportionate quantities for each raw material and
generate barcodes
    proportionate_quantities = []

    # Set the options for higher resolution
    options = {
        'module_width': 0.2,  # Default is 0.2, you can reduce it for a
finer barcode
        'module_height': 5,   # Adjust height
        'font_size': 0,       # Hide the text
        'padding': 0,         # No Padding (if the library supports it)
        'dpi': 300,           # Increase the DPI for better quality
        'write_text': False   # Don't write the text below the barcode
    }

    for index, item in enumerate(recepi_items):
        proportionate_qty = item.qty * bom_factor

        # Generate barcode for rm_cd + rm_des
        rm_code = f"{item.rm_cd.rm_cd}"
        barcode_class = barcode.get_barcode_class('code128')
        barcode_image = barcode_class(rm_code,
writer=ImageWriter())

        # Use the options while writing the barcode to the buffer
        buffer = BytesIO()
        barcode_image.write(buffer, options)  # Passing options here
        barcode_data =
base64.b64encode(buffer.getvalue()).decode('utf-8')

        # Determine if a page break is needed after every 10th item
        page_break = True if (index + 1) % 10 == 0 else False
```

```
        proportionate_quantities.append({
            'rm_cd': item.rm_cd,
            'rm_des': item.rm_cd.rm_des,
            'recepi_qty': item.qty,
            'proportionate_qty': proportionate_qty,
            'uom': item.rm_cd.uom,  # Assuming 'uom' is part of RawMat
model
            'barcode_data': barcode_data,  # Include the barcode image
data
            'page_break': page_break  # Include the page break flag
        })

    context = {
        'mixprod': mixprod,
        'recepi_items': recepi_items,
        'total_recepi_qty': total_recepi_qty,
        'bom_factor': bom_factor,
        'proportionate_quantities': proportionate_quantities,
        'total_proportionate_qty': total_proportionate_qty,
    }

    return render(request, 'product/mixprod_detail.html', context)



# Load AIML bot
kernel = aiml.Kernel()
kernel.learn("product/recepi_bot.aiml")  # Path to your AIML file

def chatbot_view(request):
    # Get user message from GET request
    user_message = request.GET.get('message')

    # If no message is provided, set a default response
    if not user_message:
        bot_response = "Please type a message."
    else:
        # Check if message contains "Show me details of Recipe"
        if "SHOW ME DETAILS OF RECIPE" in
```

```
user_message.upper():
            parts = user_message.split()
            if len(parts) >= 5:  # Ensure the message has enough parts
                recipe_code = parts[-1]
                recepi_list = Recepi.objects.filter(card_no=recipe_code)

                # If the recipe exists, format the response with details
                if recepi_list.exists():
                    bot_response = f"Recepi Card No:
{recipe_code}\nDetails:\nR.M Code\tQuantity(Kg)\n"
                    for recepi in recepi_list:
                        bot_response +=
f"{recepi.rm_cd}\t\t{recepi.qty:.3f}\n"
                else:
                    bot_response = f"Recipe {recipe_code} not found."
            else:
                bot_response = "Please provide a valid recipe code."
        else:
            # Default AIML response if message doesn't match recipe
pattern
            bot_response = kernel.respond(user_message)

    return render(request, 'product/chat.html', {'bot_response':
bot_response})


def pdf_list(request):
    # Path to the PDF directory within the media folder
    pdf_directory = os.path.join(settings.MEDIA_ROOT, 'resources',
'pdf')
    pdf_files = []

    try:
        # Generate full URLs for each file to use in the template
        pdf_files = [
            {
                'name': f,
                'url': os.path.join(settings.MEDIA_URL, 'resources', 'pdf',
f)
```

```
            }
            for f in os.listdir(pdf_directory) if f.endswith('.pdf')
        ]
    except FileNotFoundError:
        pass

    return render(request, 'product/pdf_dropdown.html', {'pdf_files':
pdf_files})

def extract_links_from_pdf(pdf_path):
    """Extracts all unique links from a PDF file."""
    doc = fitz.open(pdf_path)
    all_links = set()  # Use a set to store unique links

    for page in doc:
        for link in page.get_links():
            uri = link.get("uri", "")
            if uri:
                all_links.add(uri)  # Sets automatically remove duplicates

    return list(all_links)  # Convert back to a list before returning


def upload_pdf_extract_links(request):
    """Handles PDF file upload, extracts links, and saves the results
dynamically."""
    if request.method == "POST" and request.FILES.get("pdf_file"):
        pdf_file = request.FILES["pdf_file"]
        save_dir = os.path.join(settings.MEDIA_ROOT,
"extracted_links")
        os.makedirs(save_dir, exist_ok=True)

        # Save uploaded file temporarily
        file_path = os.path.join(save_dir, pdf_file.name)
        file_name = default_storage.save(file_path,
ContentFile(pdf_file.read()))

        # Extract links
        extracted_links =
```

```
extract_links_from_pdf(default_storage.path(file_name))

        # Save extracted links to a text file
        links_file_path = os.path.join(save_dir,
f"{pdf_file.name}_links.txt")
        with open(links_file_path, "w") as f:
            for link in extracted_links:
                f.write(link + "\n")

        return JsonResponse({
            "message": "Links extracted successfully",
            "links": extracted_links,
            "saved_file": links_file_path,
        })

    return render(request, "product/upload_pdf.html")

# Downloading file following extract_link.txt

# Disable SSL warnings (Optional, but recommended for
production)
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarnin
g)

# Headers to mimic a real browser request
HEADERS = {
    "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124
Safari/537.36",
    "Referer": "https://www.google.com",
    "Accept":
"text/html,application/pdf,application/vnd.ms-excel,application/mswo
rd,*/*",
    "Connection": "keep-alive"
}

def log_missing_links(save_dir, message, url):
    """Logs missing links (skipped HTML pages and 403 errors) to a
```

```
file."""
    log_file = os.path.join(save_dir, "missing_log.txt")
    with open(log_file, "a") as log:
        log.write(f"{message}: {url}\n")

def convert_google_drive_link(url):
    """Convert Google Drive view links to direct download links"""
    if "drive.google.com" in url and "/file/d/" in url:
        file_id = url.split("/file/d/")[1].split("/")[0]
        return
f"https://drive.google.com/uc?export=download&id={file_id}"
    return url

def get_file_extension(response, url, file_bytes):
    """Determines the correct file extension from headers, magic
bytes, or URL."""
    content_type = response.headers.get("Content-Type", "")
    content_disposition =
response.headers.get("Content-Disposition", "")

    if "filename=" in content_disposition:
        filename = content_disposition.split("filename=")[-1].strip('"')
        ext = os.path.splitext(filename)[1]
        if ext:
            return ext

    if content_type:
        if "text/html" in content_type:
            return None  # Skip downloading HTML pages
        ext = mimetypes.guess_extension(content_type)
        if ext:
            return ext

    mime = magic.Magic(mime=True)
    detected_mime = mime.from_buffer(file_bytes[:2048])
    ext = mimetypes.guess_extension(detected_mime)
    if ext:
        return ext
```

```
    url_path = urlparse(url).path
    url_ext = os.path.splitext(url_path)[1]
    if url_ext:
        return url_ext

    return ".bin"

def download_file(url, save_dir, file_index):
    """Downloads a file and assigns the correct file extension."""
    session = requests.Session()
    session.headers.update(HEADERS)

    url = convert_google_drive_link(url)

    try:
        response = session.get(url, stream=True,
allow_redirects=True, verify=False)
        response.raise_for_status()

        file_bytes = response.content[:2048]
        file_extension = get_file_extension(response, url, file_bytes)

        if file_extension is None:
            print(f"⚠ Skipping {url} (appears to be a webpage, not a
document)")
            log_missing_links(save_dir, "Skipped HTML Page", url)
            return None

        file_name = f"file_{file_index}{file_extension}"
        save_path = os.path.join(save_dir, file_name)

        with open(save_path, "wb") as file:
            file.write(response.content)

        print(f"✅ Downloaded: {save_path}")
        return file_name

    except requests.exceptions.HTTPError as e:
        if response.status_code == 403:
```

```
            print(f"✗ Forbidden: {url} - This site may require login")
            log_missing_links(save_dir, "403 Forbidden", url)
        else:
            print(f"✗ Error downloading {url}: {e}")
            log_missing_links(save_dir, "Download Error", url)
        return None

def upload_links_and_download(request):
    if request.method == "POST" and request.FILES.get("links_file"):
        links_file = request.FILES["links_file"]
        file_path = default_storage.save(f"uploads/{links_file.name}",
ContentFile(links_file.read()))

        save_dir = os.path.join("downloads",
os.path.splitext(links_file.name)[0])
        os.makedirs(save_dir, exist_ok=True)

        with default_storage.open(file_path, "r") as file:
            links = [line.strip() for line in file.readlines() if line.strip()]

        downloaded_files = []
        for index, link in enumerate(links, start=1):
            downloaded_file = download_file(link, save_dir, index)
            if downloaded_file:
                downloaded_files.append(downloaded_file)

        return JsonResponse({"status": "success", "downloaded_files":
downloaded_files, "save_dir": save_dir})

    return render(request, "product/upload_links.html")
```