**view.py**

import datetime

import os.path

import os.path

import random

import shutil

from difflib import SequenceMatcher

from os.path import join

from pathlib import Path

import boto3 as boto3

import numpy as np

import pandas as pd

import requests

from django.contrib.auth import authenticate, login, logout

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

from django.core.files.storage import FileSystemStorage

from django.http import HttpResponseForbidden, FileResponse

from django.shortcuts import render, redirect

from docxtpl import DocxTemplate

from lxml import etree

from transliterate import translit

from docx import Document as Doc

from excel\_to\_doc\_parser.models import CustomUser, Role, Document

from parser\_server.settings import BASE\_DIR, MEDIA\_ROOT

PLANE\_PATH = join(MEDIA\_ROOT, "excel", "planes", "18048 09.03.01 WEB OFO 2022.xlsx")

MATRIX\_PATH = join(MEDIA\_ROOT, "excel", "matrices", "Matrix WEB.xlsx")

TEMPLATE\_PATH = join(BASE\_DIR, "excel\_to\_doc\_parser", "py", "template.docx")

def check\_number(num):

if num % 10 == 1 and num != 11:

return '1'

elif 1 < num % 10 < 5 and (num > 19 or num < 5):

return '2'

else:

return '3'

@login\_required(login\_url='/login/')

def index(request):

context = {}

if request.user.is\_authenticated:

context = {"hello": "hello", "custom\_user": CustomUser.objects.get(user=request.user)}

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

if context["custom\_user"].role\_id == 1:

if request.method == "POST":

fs = FileSystemStorage()

folder = MEDIA\_ROOT

for filename in os.listdir(folder):

file\_path = os.path.join(folder, filename)

if filename == ".gitkeep":

continue

try:

if os.path.isfile(file\_path) or os.path.islink(file\_path):

os.unlink(file\_path)

elif os.path.isdir(file\_path):

shutil.rmtree(file\_path)

except Exception as e:

print('An error appear ' + str(e))

else:

return HttpResponseForbidden()

return render(request, "main.html", context)

def get\_current\_disciplines() -> list:

disciplines = []

data = parse\_plane(PLANE\_PATH)["Основные дисциплины"]

header = get\_header(PLANE\_PATH)

for i, row in data[header['Название дисциплины']].items():

if not ("блок" in row.lower() or "часть" in row.lower() or "дисциплины" in row.lower()):

if "\*" in row:

row = row[:row.find("\*")].strip()

disciplines.append(row)

return disciplines

def get\_profile\_name() -> str:

df = pd.read\_excel(PLANE\_PATH, header=None, index\_col=None)

data = df.dropna(axis="columns", how="all").dropna(axis="rows", how="all")

for \_, row in data.iterrows():

if "Профиль" in np.array2string(row.values):

for cell in row.values:

if isinstance(cell, str) and "Профиль:" in cell:

return cell.replace("Профиль:", "").strip()

return ""

def get\_program\_code() -> str:

df = pd.read\_excel(PLANE\_PATH, header=None, index\_col=None)

data = df.dropna(axis="columns", how="all").dropna(axis="rows", how="all")

for \_, row in data.iterrows():

if "по направлению подготовки" in np.array2string(row.values):

for cell in row.values:

if isinstance(cell, str) and "по направлению подготовки" in cell:

return cell.replace("по направлению подготовки", "").strip()

return ""

def parse\_matrix(filename) -> (dict, dict):

frame = pd.read\_excel(filename, header=None, index\_col=None)

header = {}

data = {}

all\_competencies = {}

header\_row = frame.loc[frame[0] == "КОМПЕТЕНЦИИ"]

header\_row.columns = pd.RangeIndex(header\_row.columns.size)

header\_row.reset\_index(drop=True, inplace=True)

header\_row = header\_row.iloc[0].dropna().to\_frame().T

for i, key in enumerate(header\_row.values[0], 1):

if not pd.isna(key):

header[key] = i

frame = frame.drop(range(frame.loc[frame[0] == "КОМПЕТЕНЦИИ"].head().index[0] + 1))

frame = frame.dropna(axis="columns", how="all")

frame.reset\_index(drop=True, inplace=True)

all\_comps = frame[frame.columns[:2]]

all\_comps.columns = ['competency', 'indicator']

all\_comps.reset\_index(drop=True, inplace=True)

universal\_comp\_end = \

all\_comps.loc[all\_comps['competency'] == "Общепрофессиональные компетенции и индикаторы"].head().index[0]

common\_prof\_comp\_end = \

all\_comps.loc[all\_comps['competency'].str.strip() == "Профессиональные компетенции и индикаторы"].head().index[

0]

relation\_matrix = frame[frame.columns[2:]]

relation\_matrix.drop([0, universal\_comp\_end, common\_prof\_comp\_end], axis=0, inplace=True)

relation\_matrix.reset\_index(drop=True, inplace=True)

all\_comps.drop([0, universal\_comp\_end, common\_prof\_comp\_end], axis=0, inplace=True)

all\_comps.reset\_index(drop=True, inplace=True)

all\_competencies\_names = all\_comps['competency'].dropna()

all\_competencies\_names.reset\_index(drop=True, inplace=True)

all\_indicators = all\_comps['indicator']

disciplines = header

disciplines.pop('КОМПЕТЕНЦИИ', None)

disciplines.pop("ИНДИКАТОРЫ", None)

for key, value in disciplines.items():

relation\_column = relation\_matrix.iloc[:, value - 3]

data[key] = {}

all\_competencies[key] = []

data[key]['universal\_competencies'] = []

data[key]['general\_professional\_competencies'] = []

data[key]['professional\_competencies'] = []

current\_discipline\_relation = pd.concat([all\_indicators, relation\_column], axis=1)

current\_discipline\_relation.columns = ['indicator', 'value']

current\_discipline\_relation = current\_discipline\_relation[current\_discipline\_relation['value'].notna()]

all\_comp\_for\_disc = {'competency': [], 'indicator': []}

for i, values in current\_discipline\_relation.iterrows():

if i % 3 == 2:

comp\_name\_index = i - 2

elif i % 3 == 1:

comp\_name\_index = i - 1

else:

comp\_name\_index = i

all\_comp\_for\_disc['competency'].append(all\_competencies\_names.iloc[comp\_name\_index // 3])

all\_comp\_for\_disc['indicator'].append(values['indicator'])

all\_comp\_for\_disc\_df = pd.DataFrame.from\_dict(all\_comp\_for\_disc)

all\_comp\_for\_disc\_df = all\_comp\_for\_disc\_df.groupby(['competency'])['indicator'].apply(list)

all\_comp\_for\_disc\_df = all\_comp\_for\_disc\_df.to\_frame()

for i, values in all\_comp\_for\_disc\_df.iterrows():

indicators = []

for indicator in values['indicator']:

indicators.append([

indicator.split(" ")[0],

' '.join(word for word in indicator.split(" ")[1:]).strip()

])

all\_competencies[key].append({

'competency\_code': i.split(".")[0],

'competency\_name': ' '.join(word for word in i.split(" ")[1:]).strip(),

'indicators': indicators

})

if i.startswith("УК"):

data[key]['universal\_competencies'].append({

'competency\_code': i.split(".")[0],

'competency\_name': ' '.join(word for word in i.split(" ")[1:]).strip(),

'indicators': indicators

})

elif i.startswith("ОПК"):

data[key]['general\_professional\_competencies'].append({

'competency\_code': i.split(".")[0],

'competency\_name': ' '.join(word for word in i.split(" ")[1:]).strip(),

'indicators': indicators

})

else:

data[key]['professional\_competencies'].append({

'competency\_code': i.split(".")[0],

'competency\_name': ' '.join(word for word in i.split(" ")[1:]).strip(),

'indicators': indicators

})

return data, all\_competencies

def hours\_to\_zet(z):

h = round(z / 36, 1)

if h == int(h):

return int(h)

else:

return h

def number\_to\_words(n):

less\_than\_ten = {1: 'первом', 2: 'втором', 3: 'третьем', 4: 'четвёртом',

5: 'пятом', 6: 'шестом', 7: 'седьмом', 8: 'восьмом',

9: 'девятом'}

ten = {10: 'десятом'}

from\_eleven\_to\_nineteen = {11: 'одиннадцатом', 12: 'двенадцатом',

13: 'тринадцатом', 14: 'четырнадцатом',

15: 'пятнадцатом', 16: 'шестнадцатом',

17: 'семнадцатом', 18: 'восемнадцатом',

19: 'девятнадцатом'}

n1 = n % 10

n2 = n - n1

if n < 10:

return less\_than\_ten.get(n)

elif 10 < n < 20:

return from\_eleven\_to\_nineteen.get(n)

elif n >= 10 and n in ten:

return ten.get(n)

else:

return ten.get(n2) + ' ' + less\_than\_ten.get(n1)

@login\_required(login\_url='/login/')

def documents(request):

context = {"documents": Document.objects.filter(user\_id=request.user.id),

"custom\_user": CustomUser.objects.get(user=request.user), "disciplines": get\_current\_disciplines()}

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

if request.method == "POST":

if request.POST.get("generate"):

folder = join(str(BASE\_DIR), "excel\_to\_doc\_parser/media/generated\_files/docx")

data\_df = parse\_plane(PLANE\_PATH)["Основные дисциплины"]

header = get\_header(PLANE\_PATH)

hours = {}

for i, row in data\_df[header['Название дисциплины']].items():

if not ("блок" in row.lower() or "часть" in row.lower() or "дисциплины" in row.lower()):

if "\*" in row:

row = row[:row.find("\*")].strip()

hours[row] = {}

hours[row]["lections"] = []

hours[row]["seminars"] = []

hours[row]["labs"] = []

hours[row]["srs"] = []

hours[row]["exam"] = []

hours[row]["test"] = []

if not pd.isna(data\_df.iloc[i - 1][header["Лекции"]]):

hours[row]["lections"].append(data\_df.iloc[i - 1][header["Лекции"]])

if not pd.isna(data\_df.iloc[i - 1][header["Семинары и практические занятия"]]):

hours[row]["seminars"].append(data\_df.iloc[i - 1][header["Семинары и практические занятия"]])

if not pd.isna(data\_df.iloc[i - 1][header["Лабораторные работы"]]):

hours[row]["labs"].append(data\_df.iloc[i - 1][header["Лабораторные работы"]])

if not pd.isna(data\_df.iloc[i - 1][header["СРС"]]):

hours[row]["srs"].append(data\_df.iloc[i - 1][header["СРС"]])

if not pd.isna(data\_df.iloc[i - 1][header["Экзамены"]]):

hours[row]["exam"].append(data\_df.iloc[i - 1][header["Экзамены"]])

if not pd.isna(data\_df.iloc[i - 1][header["Зачёты"]]):

hours[row]["test"].append(data\_df.iloc[i - 1][header["Зачёты"]])

for filename in os.listdir(folder):

file\_path = os.path.join(folder, filename)

if filename == ".gitkeep":

continue

try:

if os.path.isfile(file\_path) or os.path.islink(file\_path):

os.unlink(file\_path)

elif os.path.isdir(file\_path):

shutil.rmtree(file\_path)

except Exception as e:

print('An error appear ' + str(e))

data, all\_competencies = parse\_matrix(MATRIX\_PATH)

discipline = Document.objects.get(pk=request.POST.get('document')).document\_name

try:

context\_plane\_df = data\_df[data\_df[header['Название дисциплины']] == discipline]

except KeyError:

for error\_key in data\_df[header['Название дисциплины']]:

if SequenceMatcher(None, discipline, error\_key).ratio() >= 0.75:

context\_plane\_df = data\_df[data\_df[header['Название дисциплины']] == error\_key]

break

context\_plane = {

'intensity\_ZET': int(context\_plane\_df[header['Всего, ЗЕТ']].values[0]),

'intensity\_hours': int(context\_plane\_df[header['ВСЕГО по структуре']].values[0]),

'total\_homework\_hours': int(context\_plane\_df[header['СРС']].values[0]), 'courses': []

}

context\_plane['intensity\_ZET\_check'] = check\_number(context\_plane['intensity\_ZET'])

context\_plane['intensity\_hours\_check'] = check\_number(context\_plane['intensity\_hours'])

context\_plane['total\_homework\_hours\_check'] = check\_number(context\_plane['total\_homework\_hours'])

for key in header:

if "семестр" in key:

if not pd.isna(context\_plane\_df[header[key]].values[0]):

context\_plane['courses'].append({

'ZET': hours\_to\_zet(int(context\_plane\_df[header[key]].values[0]) + int(

context\_plane\_df[header['СРС']].values[0])),

'hours': int(context\_plane\_df[header[key]].values[0]) + int(

context\_plane\_df[header['СРС']].values[0]),

'homework\_time': int(context\_plane\_df[header['СРС']].values[0]),

'semester': number\_to\_words(int(key.split(" ")[0])),

'course': number\_to\_words(int(round(int(key.split(" ")[0]) / 2 + 0.1))),

'exam': context\_plane\_df[header['Экзамены']].values[0] if not pd.isna(

context\_plane\_df[header['Экзамены']].values[0]) and key.split(" ")[0] in context\_plane\_df[header['Экзамены']].values[0] else "",

'test': context\_plane\_df[header['Зачёты']].values[0] if not pd.isna(

context\_plane\_df[header['Зачёты']].values[0]) and key.split(" ")[0] in context\_plane\_df[header['Зачёты']].values[0] else ""

})

for i, \_ in enumerate(context\_plane['courses']):

context\_plane['courses'][i]['ZET\_check'] = check\_number(context\_plane['courses'][i]['ZET'])

context\_plane['courses'][i]['hours\_check'] = check\_number(context\_plane['courses'][i]['hours'])

context\_plane['courses'][i]['homework\_time\_check'] = check\_number(

context\_plane['courses'][i]['homework\_time'])

try:

context\_plane["hours"] = hours[discipline]

except KeyError:

for error\_key in data\_df[header['Название дисциплины']]:

if SequenceMatcher(None, discipline, error\_key).ratio() >= 0.75:

context\_plane["hours"] = hours[error\_key]

break

doc = DocxTemplate(TEMPLATE\_PATH)

data = dict(data[discipline], \*\*xml\_parser(request))

data['all\_comp'] = all\_competencies[discipline]

data['dean'] = "Д.Г. Демидов"

data['head\_of\_faculty'] = "Е.В. Пухова"

data['rop'] = "М.В. Даньшина"

data["program\_name"] = discipline

data["current\_year"] = datetime.date.today().year

data["program\_code"] = Document.objects.get(pk=request.POST.get('document')).program\_code

data["profile\_name"] = Document.objects.get(pk=request.POST.get('document')).profile\_name

data["year\_start"] = data['current\_year']

doc.render(dict(data, \*\*context\_plane))

for i in range(len(doc.tables)):

table = doc.tables[i].\_tbl

for row in doc.tables[i].rows:

if len(row.cells) and len(row.cells[0].text.strip()) == 0 and len(set(row.cells)) == 1:

table.remove(row.\_tr)

discipline = translit(discipline, "ru", reversed=True)

doc.save(join(str(BASE\_DIR), folder, "{}.docx".format(discipline)))

context['path'] = join(folder, "{}.docx".format(discipline))

context['name'] = discipline + '.docx'

return redirect("/download/?file={}&name=".format(context['path'], context["name"]))

program\_name = request.POST.get("program\_name")

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

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

user = request.user.id

new\_document = Document(link\_to\_xml\_id=link, status\_id=status, user\_id=user, document\_name=program\_name,

profile\_name=get\_profile\_name(), program\_code=get\_program\_code())

new\_document.save()

return redirect('/documents')

return render(request, "./docx\_creation/document.html", context)

def main\_info\_parser(part, content):

data = {}

for field in part:

data[str(field.tag)] = field.text

content["main\_info"] = data

return content

def files\_parser(part, content):

data = {}

for field in part:

data[str(field.tag)] = field.text

content["files"] = data

return content

def targets\_parser(part, content):

data = []

for field in part:

if not field.text:

part.remove(field)

for i, field in enumerate(part):

data.append(field.text.strip() + (";" if i < len(part) - 1 else ""))

content["targets"] = data

return content

def tasks\_parser(part, content):

data = []

for field in part:

if not field.text:

part.remove(field)

for i, field in enumerate(part):

data.append(field.text.strip() + (";" if i < len(part) - 1 else ""))

content["tasks"] = data

return content

def sections\_parser(part, content):

for i, field in enumerate(part):

content[str(field[0].text)] = field[1].text

content["section"] = content

return content

def disciplines\_parser(part, content):

data = []

for field in part:

if not field.text:

part.remove(field)

for i, field in enumerate(part):

if field.text:

data.append(field.text.strip() + (";" if field != part[-1] else "."))

content["disciplines"] = data

return content

def sections\_content\_parser(part, content):

data = []

for i, field in enumerate(part):

if field[0].text != "#TODO":

data.append([field[0].text, int(field[1].text), int(field[2].text), int(field[3].text), int(field[4].text),

field[5].text])

content["sections"] = data

return content

def marks\_parser(part, content):

data = []

content[part[0].tag] = part[0].text

content[part[1].tag] = part[1].text

content[part[2].tag] = part[2].text

content[part[3].tag] = part[3].text

# for i, field in enumerate(part[4]):

# content["{}{}".format(field.tag, i)] = field.text

for i, field in enumerate(part[4]):

if field[0].text:

data.append([field[0].text, field[1].text])

content["marks"] = data

return content

def literature\_parser(part, content):

data = {}

for j in range(len(part)):

books = []

for i, field in enumerate(part[j]):

if field.text:

books.append(field.text.strip())

data[part[j].tag] = books

content["literature"] = data

return content

def software\_parser(part, content):

data = []

for field in part:

if not field.text:

part.remove(field)

for i, field in enumerate(part):

data.append(field.text.strip())

content["software"] = data

return content

def evaluation\_tools\_parser(part, content):

for i, field in enumerate(part):

content["{}{}".format(field.tag, i)] = field.text

content["evaluation\_tool"] = content

return content

def tasks\_for\_students\_parser(part, content):

for i, field in enumerate(part):

content["{}{}".format(field.tag, i)] = field.text

content["tasks\_for\_students"] = content

return content

def education\_technologies\_parser(part, content):

data = {}

education\_technologies\_in = []

education\_technologies\_out = []

for field in part[0]:

if not field.text:

part[0].remove(field)

for field in part[1]:

if not field.text:

part[1].remove(field)

for i, field in enumerate(part[0]):

education\_technologies\_in.append(field.text.strip() + (";" if field != part[-1] else "."))

data["education\_technologies\_in"] = education\_technologies\_in

for i, field in enumerate(part[1]):

education\_technologies\_out.append(field.text.strip() + (";" if field != part[-1] else "."))

data["education\_technologies\_out"] = education\_technologies\_out

content["education\_technologies"] = data

return content

def mark\_criteries\_parser(part, content):

data = []

for field in part:

if not field.text:

part.remove(field)

for i, field in enumerate(part):

data.append(field.text.strip() + (";" if field != part[-1] else "."))

content["mark\_criteries"] = data

return content

def fos\_parser(part, content):

data = {

'kr': {},

'course\_work': {},

'exam\_questions': {},

'test\_questions': [],

'example\_exam\_questions': [],

'example\_exam\_task': '',

'os': []

}

data['exam\_questions']['theory'] = []

data['exam\_questions']['tasks'] = []

data['course\_work']['themes'] = []

data['course\_work']['contents'] = []

for field in part:

if field.getchildren():

if field.tag == "kr":

for i, questions in enumerate(field.getchildren()):

data['kr'][i] = []

for question in questions:

data['kr'][i].append(question.text)

elif field.tag == 'course\_work':

for theme in field.getchildren()[0]:

data['course\_work']['themes'].append(theme.text)

for content in field.getchildren()[1]:

data['course\_work']['contents'].append(content.text)

elif field.tag == 'exam\_questions':

for question in field.getchildren()[0]:

data['exam\_questions']['theory'].append(question.text)

for tasks in field.getchildren()[1]:

data['exam\_questions']['tasks'].append(tasks.text)

elif field.tag == 'os\_list':

for os\_list in field.getchildren():

data['os'].append([child.text for child in os\_list.getchildren()])

else:

for question in field:

data['test\_questions'].append(question.text)

if field.tag == 'all\_os':

data['all\_os'] = field.text

for key in data:

data[key] = {k: v for k, v in data[key].items() if v} if isinstance(data[key], dict) else data[key]

if data['exam\_questions']:

data['example\_exam\_questions'] = random.sample(data['exam\_questions']['theory'], 2)

data['example\_exam\_task'] = random.sample(data['exam\_questions']['tasks'], 1)

content['fos'] = data

return content

def xml\_parser(request) -> dict:

root = etree.fromstring(

requests.get(download\_xml\_from\_s3(

request,

translit("{}.xml".format(

Document.objects.get(

pk=request.POST.get("document")

).document\_name).replace(" ", "\_"), "ru", reversed=True))

).content

)

content = {}

functions = {

"main\_info": main\_info\_parser,

"files": files\_parser,

"targets": targets\_parser,

"tasks": tasks\_parser,

"sections": sections\_parser,

"disciplines": disciplines\_parser,

"sections\_content": sections\_content\_parser,

"marks": marks\_parser,

"literature": literature\_parser,

"software": software\_parser,

"evaluation\_tools": evaluation\_tools\_parser,

"tasks\_for\_students": tasks\_for\_students\_parser,

"education\_technologies": education\_technologies\_parser,

"mark\_criteries": mark\_criteries\_parser,

"fos": fos\_parser

}

for part in root:

content = functions.get(part.tag, lambda: "Invalid tag")(part, content)

return content

@login\_required(login\_url='/login/')

def themes(request):

context = {}

if request.user.is\_authenticated:

context["custom\_user"] = CustomUser.objects.get(user=request.user)

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

if request.method == "GET":

context["document"] = request.GET.get("document")

if request.method == "POST":

context["document"] = request.POST.get("document")

context["discipline"] = Document.objects.get(pk=context["document"]).document\_name

context["profile"] = Document.objects.get(pk=context["document"]).profile\_name

context["status"] = Document.objects.get(pk=context["document"]).status.status

return render(request, "./docx\_creation/theme.html", context)

@login\_required(login\_url='/login/')

def document\_information(request):

context = {}

if request.user.is\_authenticated:

context["custom\_user"] = CustomUser.objects.get(user=request.user)

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

context["all\_themes"] = get\_current\_disciplines()

context["predefined\_techs\_in\_class"] = {"default": ["выполнение лабораторных работ в лабораториях вуза",

"индивидуальные и групповые консультации студентов преподавателем, в том числе в виде защиты выполненных заданий в рамках самостоятельной работы"],

"optional": [

"посещение профильных конференций и работа на мастер-классах экспертов и специалистов индустрии"]}

context["predefined\_techs\_out\_class"] = {

"default": ["подготовки к выполнению и подготовки к защите лабораторных работ",

"подготовки к текущей аттестации",

"подготовки к промежуточной аттестации"],

"optional": ["чтения литературы и освоения дополнительного материала в рамках тематики дисциплины"]}

context["predefined\_criteries\_in\_methods\_for\_students"] = {

"default": ["уровень освоения студентом учебного материала",

"умения студента использовать теоретические знания при выполнении практических задач",

"сформированность компетенций",

"оформление материала в соответствии с требованиями"],

"optional": []}

document = Doc(join(BASE\_DIR, "excel\_to\_doc\_parser", "py", "Primerny\_perechen\_otsenochnykh\_sredstv.docx"))

table = document.tables[0]

context['os\_list'] = []

for row in table.rows[1:]:

context['os\_list'].append({

'name': row.cells[0].text,

'description': row.cells[1].text,

'short\_name': row.cells[2].text

})

data = parse\_plane(PLANE\_PATH)["Основные дисциплины"]

header = get\_header(PLANE\_PATH)

hours = {}

for i, row in data[header['Название дисциплины']].items():

if not ("блок" in row.lower() or "часть" in row.lower() or "дисциплины" in row.lower()):

if "\*" in row:

row = row[:row.find("\*")].strip()

hours[row] = {}

hours[row]["lections"] = []

hours[row]["seminars"] = []

hours[row]["labs"] = []

hours[row]["srs"] = []

hours[row]["exam"] = []

hours[row]["course\_work"] = []

if not pd.isna(data.iloc[i - 1][header["Лекции"]]):

hours[row]["lections"].append(data.iloc[i - 1][header["Лекции"]])

if not pd.isna(data.iloc[i - 1][header["Семинары и практические занятия"]]):

hours[row]["seminars"].append(data.iloc[i - 1][header["Семинары и практические занятия"]])

if not pd.isna(data.iloc[i - 1][header["Лабораторные работы"]]):

hours[row]["labs"].append(data.iloc[i - 1][header["Лабораторные работы"]])

if not pd.isna(data.iloc[i - 1][header["СРС"]]):

hours[row]["srs"].append(data.iloc[i - 1][header["СРС"]])

if not pd.isna(data.iloc[i - 1][header["Экзамены"]]):

hours[row]["exam"].append(data.iloc[i - 1][header["Экзамены"]])

if not pd.isna(data.iloc[i - 1][header["Курсовые работы"]]):

hours[row]["course\_work"].append(data.iloc[i - 1][header["Курсовые работы"]])

if not pd.isna(data.iloc[i - 1][header["Курсовые проекты"]]):

hours[row]["course\_work"].append(data.iloc[i - 1][header["Курсовые проекты"]])

if request.method == "GET":

context["document"] = request.GET.get("document")

context["theme"] = Document.objects.get(pk=request.GET.get("document")).document\_name

if request.method == "POST":

# context["last\_values"] = xml\_parser(request)

context["document"] = request.POST.get("document")

context["theme"] = Document.objects.get(pk=request.POST.get("document")).document\_name

context["hours"] = hours[context["theme"]]

return render(request, "./docx\_creation/targets.html", context)

def generate\_xml(request):

root = etree.Element("root")

tree = etree.ElementTree(root)

main\_info = etree.Element("main\_info")

desc = etree.Element("discipline")

desc.text = Document.objects.get(pk=request.POST.get("document")).document\_name

main\_info.append(desc)

prof = etree.Element("profile")

prof.text = Document.objects.get(pk=request.POST.get("document")).profile\_name

main\_info.append(prof)

course = etree.Element("course")

course.text = "#TODO"

main\_info.append(course)

status = etree.Element("status")

status.text = Document.objects.get(pk=request.POST.get("document")).status.status

main\_info.append(status)

elective = etree.Element("elective")

elective.text = "#TODO"

main\_info.append(elective)

root.append(main\_info)

files = etree.Element("files")

files\_list = ["rpd", "annotation", "fos", "method", "review", "plan", "matrix", "program"]

try:

for element in files\_list:

file = etree.Element(element)

file.text = "#TODO"

files.append(file)

except Exception as exc:

print(exc)

root.append(files)

targets = etree.Element("targets")

try:

for element in request.POST.get("targets").split(";"):

target = etree.Element("target")

target.text = element

targets.append(target)

except Exception as exc:

print(exc)

root.append(targets)

tasks = etree.Element("tasks")

try:

for element in request.POST.get("tasks").split(";"):

task = etree.Element("task")

task.text = element

tasks.append(task)

except Exception as exc:

print(exc)

root.append(tasks)

sections = etree.Element("sections")

try:

for element in request.POST.get("all\_sections").split(";"):

if element:

data = element.split(":")

section = etree.Element("section")

section\_name = etree.Element("section\_name")

section\_name.text = data[0]

section.append(section\_name)

hours\_lections = etree.Element("hours\_lections")

hours\_lections.text = data[1]

section.append(hours\_lections)

hours\_labs = etree.Element("hours\_labs")

hours\_labs.text = data[2]

section.append(hours\_labs)

hours\_seminars = etree.Element("hours\_seminars")

hours\_seminars.text = data[3]

section.append(hours\_seminars)

hours\_srs = etree.Element("hours\_srs")

hours\_srs.text = data[4]

section.append(hours\_srs)

sections.append(section)

except Exception as exc:

section = etree.Element("section")

section\_name = etree.Element("section\_name")

section\_name.text = "#TODO"

section.append(section\_name)

hours = etree.Element("hours")

hours.text = "#TODO"

section.append(hours)

sections.append(section)

raise exc

root.append(sections)

disciplines = etree.Element("disciplines")

try:

for element in request.POST.get("all\_modules").split(";"):

discipline = etree.Element("discipline")

discipline.text = element

disciplines.append(discipline)

except Exception as exc:

discipline = etree.Element("discipline")

discipline.text = "#TODO"

disciplines.append(discipline)

print(exc)

root.append(disciplines)

sections\_content = etree.Element("sections\_content")

try:

for element in request.POST.get("all\_sections").split(";"):

data = element.split(":")

section\_content = etree.Element("section\_content")

theme = etree.Element("theme")

theme.text = data[0]

section\_content.append(theme)

hours\_lections = etree.Element("hours\_lections")

hours\_lections.text = data[1]

section\_content.append(hours\_lections)

hours\_labs = etree.Element("hours\_labs")

hours\_labs.text = data[2]

section\_content.append(hours\_labs)

hours\_seminars = etree.Element("hours\_seminars")

hours\_seminars.text = data[3]

section\_content.append(hours\_seminars)

hours\_srs = etree.Element("hours\_srs")

hours\_srs.text = data[4]

section\_content.append(hours\_srs)

content = etree.Element("content")

content.text = data[-1]

section\_content.append(content)

sections\_content.append(section\_content)

except Exception as exc:

section\_content = etree.Element("section\_content")

theme = etree.Element("theme")

theme.text = "#TODO"

section\_content.append(theme)

hours = etree.Element("hours")

hours.text = "#TODO"

section\_content.append(hours)

content = etree.Element("content")

content.text = "#TODO"

section\_content.append(content)

sections\_content.append(section\_content)

print(exc)

root.append(sections\_content)

marks = etree.Element("marks")

competency = etree.Element("competency")

competency.text = request.POST.get("competentions")

marks.append(competency)

attestation = etree.Element("attestation")

attestation.text = request.POST.get("attestation")

marks.append(attestation)

brs = etree.Element("brs")

brs.text = request.POST.get("score\_system")

marks.append(brs)

brs\_description = etree.Element("brs\_description")

brs\_description.text = request.POST.get("score\_system\_desc")

marks.append(brs\_description)

# competencies = etree.Element("competencies")

# try:

# for element in request.POST.get("competencies").split(";") or "":

# competency = etree.Element("theme")

# competency.text = "#TODO"

# competencies.append(competency)

# except Exception as exc:

# competency = etree.Element("theme")

# competency.text = "#TODO"

# competencies.append(competency)

# print(exc)

# marks.append(competencies)

intermediate = etree.Element("intermediate")

try:

for element in request.POST.get("all\_marks").split(";"):

data = element.split(":")

mark = etree.Element("mark")

value = etree.Element("value")

value.text = data[0]

mark.append(value)

characteristics = etree.Element("characteristics")

characteristics.text = data[1]

mark.append(characteristics)

intermediate.append(mark)

except Exception as exc:

mark = etree.Element("mark")

value = etree.Element("value")

competency.text = "#TODO"

mark.append(value)

characteristics = etree.Element("characteristics")

characteristics.text = "#TODO"

mark.append(characteristics)

intermediate.append(mark)

print(exc)

marks.append(intermediate)

root.append(marks)

literature = etree.Element("literature")

main = etree.Element("main")

try:

for element in request.POST.get("main\_lit").split(";"):

book = etree.Element("book")

book.text = element

main.append(book)

except Exception as exc:

book = etree.Element("book")

book.text = "#TODO"

main.append(book)

print(exc)

literature.append(main)

additional = etree.Element("additional")

try:

for element in request.POST.get("extra\_lit").split(";"):

book = etree.Element("book")

book.text = element

additional.append(book)

except Exception as exc:

book = etree.Element("book")

book.text = "#TODO"

additional.append(book)

print(exc)

literature.append(additional)

digital = etree.Element("digital")

try:

for element in request.POST.get("digital\_lit").split(";") or "":

resources = etree.Element("resources")

resources.text = element

digital.append(resources)

except Exception as exc:

resources = etree.Element("resources")

resources.text = "#TODO"

digital.append(resources)

print(exc)

literature.append(digital)

root.append(literature)

software = etree.Element("software")

try:

for element in request.POST.get("software").split(";"):

program = etree.Element("program")

program.text = element

software.append(program)

except Exception as exc:

program = etree.Element("program")

program.text = "#TODO"

software.append(program)

print(exc)

root.append(software)

evaluation\_tools = etree.Element("evaluation\_tools")

try:

for element in request.POST.get("evaluation\_tools").split(";"):

tool = etree.Element("tool")

tool.text = element

evaluation\_tools.append(tool)

except Exception as exc:

tool = etree.Element("tool")

tool.text = "#TODO"

evaluation\_tools.append(tool)

print(exc)

root.append(evaluation\_tools)

tasks\_for\_students = etree.Element("tasks\_for\_students")

try:

for element in request.POST.get("tasks\_from\_file").split(";") or "":

task = etree.Element("task")

task.text = element

tasks\_for\_students.append(task)

except Exception as exc:

task = etree.Element("task")

task.text = "#TODO"

tasks\_for\_students.append(task)

print(exc)

root.append(tasks\_for\_students)

education\_technologies = etree.Element("education\_technologies")

education\_technologies\_in = etree.Element("education\_technologies\_in")

education\_technologies\_out = etree.Element("education\_technologies\_out")

try:

for element in request.POST.getlist("default\_tech\_in\_class"):

tech = etree.Element("tech")

tech.text = element

education\_technologies\_in.append(tech)

for element in request.POST.getlist("optional\_tech\_in\_class"):

tech = etree.Element("tech")

tech.text = element

education\_technologies\_in.append(tech)

for element in request.POST.getlist("default\_tech\_out\_class"):

tech = etree.Element("tech")

tech.text = element

education\_technologies\_out.append(tech)

for element in request.POST.getlist("optional\_tech\_out\_class"):

tech = etree.Element("tech")

tech.text = element

education\_technologies\_out.append(tech)

except Exception as exc:

print(exc)

education\_technologies.append(education\_technologies\_in)

education\_technologies.append(education\_technologies\_out)

root.append(education\_technologies)

mark\_criteries = etree.Element("mark\_criteries")

try:

for element in request.POST.getlist("default\_tech\_in\_class"):

tech = etree.Element("tech")

tech.text = element

mark\_criteries.append(tech)

except Exception as exc:

print(exc)

root.append(mark\_criteries)

fos = etree.Element("fos")

kr = etree.Element("kr")

course\_work = etree.Element("course\_work")

exam\_questions = etree.Element("exam\_questions")

test\_questions = etree.Element("test\_questions")

os\_list = etree.Element("os\_list")

all\_os = etree.Element("all\_os")

try:

if request.POST.getlist('kr'):

for element in request.POST.getlist("kr"):

questions = etree.Element("questions")

for quest in element.split(";"):

question = etree.Element("question")

question.text = quest

questions.append(question)

kr.append(questions)

if request.POST.get("course\_work\_themes"):

cw\_themes = etree.Element("themes")

for element in request.POST.get("course\_work\_themes").split(";"):

theme = etree.Element("theme")

theme.text = element

cw\_themes.append(theme)

course\_work.append(cw\_themes)

if request.POST.get("course\_work\_contents"):

cw\_contents = etree.Element("contents")

for element in request.POST.get("course\_work\_contents").split(";"):

content = etree.Element("content")

content.text = element

cw\_contents.append(content)

course\_work.append(cw\_contents)

if request.POST.get("exam\_questions\_theory"):

eq\_theory = etree.Element("theory")

for element in request.POST.get("exam\_questions\_theory").split(";"):

question = etree.Element("question")

question.text = element

eq\_theory.append(question)

exam\_questions.append(eq\_theory)

if request.POST.get("exam\_questions\_tasks"):

eq\_tasks = etree.Element("tasks")

for element in request.POST.get("exam\_questions\_tasks").split(";"):

task = etree.Element("task")

task.text = element

eq\_tasks.append(task)

exam\_questions.append(eq\_tasks)

if request.POST.get("test\_questions"):

for element in request.POST.get("test\_questions").split(";"):

question = etree.Element("question")

question.text = element

test\_questions.append(question)

if request.POST.get('os'):

for element in request.POST.get('os').split(";;"):

if element:

os\_element = etree.Element("os")

name = etree.Element("name")

name.text = element.split("::")[0].strip()

os\_element.append(name)

description = etree.Element("description")

description.text = element.split("::")[1].strip()

os\_element.append(description)

os\_content = etree.Element("content")

os\_content.text = element.split("::")[2].strip()

os\_element.append(os\_content)

os\_list.append(os\_element)

if request.POST.get('all\_os'):

all\_os.text = request.POST.get("all\_os").strip()[:-1]

except Exception as exc:

print(exc)

fos.append(kr)

fos.append(course\_work)

fos.append(exam\_questions)

fos.append(test\_questions)

fos.append(os\_list)

fos.append(all\_os)

root.append(fos)

path\_to\_save = join(str(BASE\_DIR), "excel\_to\_doc\_parser\\media\\generated\_files\\xml\\{}".format(request.user.id))

Path(path\_to\_save).mkdir(parents=True, exist\_ok=True)

# filename = "{}-{}.xml".format(Document.objects.get(pk=request.POST.get("document")).program\_name.program\_name,

# datetime.date.today().strftime("%m.%d.%Y"))

filename = translit(

"{}.xml".format(Document.objects.get(pk=request.POST.get("document")).document\_name).replace(" ", "\_"),

"ru", reversed=True)

tree.write(join(str(BASE\_DIR), path\_to\_save, filename), encoding="UTF-8", xml\_declaration=True, pretty\_print=True)

upload\_xml\_to\_s3(request, filename, path\_to\_save)

os.remove(join(str(BASE\_DIR), path\_to\_save, filename))

def upload\_xml\_to\_s3(request, filename, filepath):

session = boto3.session.Session()

s3 = session.client(

service\_name='s3',

endpoint\_url='https://storage.yandexcloud.net',

aws\_access\_key\_id=os.environ.get('S3\_ACCESS\_KEY'),

aws\_secret\_access\_key=os.environ.get('S3\_SECRET\_KEY'),

)

with open(join(str(BASE\_DIR), filepath, filename), "rb") as xml:

s3.put\_object(Bucket=os.environ.get('BUCKET\_NAME'), Key='xml/{}/{}'.format(request.user.id, filename),

Body=xml.read().decode("UTF-8"))

def download\_xml\_from\_s3(request, filename):

session = boto3.session.Session()

s3 = session.client(

service\_name='s3',

endpoint\_url='https://storage.yandexcloud.net',

aws\_access\_key\_id=os.environ.get('S3\_ACCESS\_KEY'),

aws\_secret\_access\_key=os.environ.get('S3\_SECRET\_KEY'),

)

return s3.generate\_presigned\_url('get\_object', Params={

'Bucket': os.environ.get('BUCKET\_NAME'),

'Key': 'xml/{}/{}'.format(request.user.id, filename)},

ExpiresIn=60)

@login\_required(login\_url='/login/')

def result(request):

context = {}

if request.user.is\_authenticated:

context["custom\_user"] = CustomUser.objects.get(user=request.user)

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

context["document"] = request.POST.get("document")

if request.method == "POST":

if request.POST.get("end") == "generate":

generate\_xml(request)

if request.POST.get("save"):

pass

return render(request, "./docx\_creation/result.html", context)

def download(request):

file = join(str(BASE\_DIR) + "/", request.GET.get('file'))

response = FileResponse(open(file, 'rb'), as\_attachment=True,

content\_type='application/vnd.openxmlformats-officedocument.wordprocessingml.document')

response['Content-Length'] = os.path.getsize(file)

return response

def login\_view(request):

if request.user.is\_authenticated:

return redirect("/documents/")

if request.method == "POST":

username = request.POST.get('login')

password = request.POST.get('password')

user = authenticate(request, username=username, password=password)

if user is not None:

login(request, user)

return redirect("/documents/")

else:

print("Error")

return render(request, "authorization.html")

def logout\_view(request):

logout(request)

if not request.user.is\_authenticated:

return redirect("/")

return render(request, "authorization.html")

@login\_required(login\_url='/login/')

def info(request):

context = {}

if request.user.is\_authenticated:

context["custom\_user"] = CustomUser.objects.get(user=request.user)

context["role"] = Role.objects.get(pk=context["custom\_user"].role\_id)

return render(request, "feedback.html", context)

def get\_disciplines\_hours(data, header, keys):

for key in keys:

for index, row in data[header[key]].items():

if not pd.isna(row):

keys[key].append((

data.iloc[index - 1].iloc[header["Название дисциплины"] - 1],

data.iloc[index - 1].iloc[header[key] - 1]))

return keys

def get\_sem(data, index):

context = {}

for i, key in enumerate(data.iloc[:, index:].values[1], index + 1):

if not pd.isna(key):

context[key] = i

else:

break

return context

def get\_hours(data, index):

context = {}

for i, key in enumerate(data.iloc[:, index:].values[1], index + 1):

if not pd.isna(key) and "курс" not in key:

context[key] = i

else:

break

return context

def get\_courses(data, index):

context = {}

for i, key in enumerate(data.iloc[:, index:].values[2], index + 1):

if not pd.isna(key) and "курс" not in key:

context[key] = i

else:

break

return context

def get\_header(filename):

header = {}

df = pd.read\_excel(filename, header=None, index\_col=None)

data = df.dropna(axis="columns", how="all")

disciplines = data.copy()

header\_row = disciplines[disciplines.loc[disciplines[2] == "Шифр"].head().index[0]:

disciplines.loc[disciplines[2] == "Шифр"].head().index[0] + 3].dropna(axis="columns",

how='all')

header\_row.columns = pd.RangeIndex(header\_row.columns.size)

for i, key in enumerate(header\_row.values[0], 1):

if not pd.isna(key):

if "распределение по семестрам" in key.lower():

header = dict(\*\*header, \*\*get\_sem(header\_row, i - 1))

elif "часы" in key.lower():

header = dict(\*\*header, \*\*get\_hours(header\_row, i - 1))

elif "распределение по курсам" in key.lower():

header = dict(\*\*header, \*\*get\_courses(header\_row, i - 1))

else:

header[key] = i

return header

def parse\_plane(filename) -> dict:

context = {}

df = pd.read\_excel(filename, header=None, index\_col=None)

data = df.dropna(axis="columns", how="all")

disciplines = data.copy()

start\_index = 0

for index, column in disciplines.items():

if "Блок 1. Дисциплины (модули)" in np.array2string(column.values):

start\_index = index

disciplines = disciplines.drop(

range(data.loc[data[start\_index] == "Блок 1. Дисциплины (модули)"].head().index[0] - 1))

context["Факультативные дисциплины"] = data[3].iloc[

range(data.loc[data[2].isin(["№ п/п"])].head().index[0], data.iloc[-1:].head().index[0] + 1)]

disciplines = disciplines.drop(

range(data.loc[data[2].isin(["№ п/п"])].head().index[0], data.iloc[-1:].head().index[0] + 1))

for column in data:

if data[column].isin(["8 семестр\n6 недель"]).any():

break

disciplines = disciplines.iloc[:, range(column - 1)]

disciplines = disciplines.dropna(axis='columns', how="all").dropna(axis="rows", how="all")

# context["Факультативные дисциплины"] = context["Факультативные дисциплины"].dropna(axis='columns', how='all').dropna(axiss='rows', how='all')

disciplines.reset\_index(drop=True, inplace=True)

new\_header = disciplines.iloc[0].astype(int)

disciplines = disciplines[1:]

disciplines.columns = new\_header

context["Основные дисциплины"] = disciplines

return context

**models.py**

from django.contrib.auth.models import User

from django.db import models

class Role(models.Model):

ADMIN = 1

DEAN = 2

HOD = 3

HEP = 4

TEACHER = 5

ROLE\_CHOICES = (

(ADMIN, 'Admin'),

(DEAN, 'Dean'),

(HOD, 'HOD'),

(HEP, 'HEP'),

(TEACHER, 'Teacher')

)

role\_type = models.CharField(choices=ROLE\_CHOICES, max\_length=128)

class CustomUser(models.Model):

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

first\_name = models.CharField(max\_length=128)

last\_name = models.CharField(max\_length=128)

second\_name = models.CharField(max\_length=128)

role = models.ForeignKey(Role, on\_delete=models.CASCADE)

class Link(models.Model):

link = models.CharField(max\_length=256)

class Status(models.Model):

STATUSES = [("В архиве", "archive"), ("Актуальный", "actual"), ("Отправлен на доработку", "revise"),

("Отклонён", "rejected"), ("В процессе составления", "making"),

("В процессе редакции", "redaction"), ("Составлен", "made"), ("Согласован", "conformed"),

("Утверждён", "approved")]

status = models.CharField(choices=STATUSES, default=STATUSES[4], max\_length=256)

class Document(models.Model):

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

status = models.ForeignKey(Status, on\_delete=models.CASCADE)

link\_to\_xml = models.ForeignKey(Link, on\_delete=models.CASCADE)

document\_name = models.CharField(max\_length=512, default="")

profile\_name = models.CharField(max\_length=512, default="")

program\_code = models.CharField(max\_length=512, default="")