**docx\_generator.py**

*from* difflib *import* SequenceMatcher  
  
*from* docxtpl *import* DocxTemplate  
  
*from* excel\_to\_doc\_parser.py.parser *import* get\_info\_from\_excel  
*from* excel\_to\_doc\_parser.py.parser\_plane *import* get\_info\_from\_education\_plane  
  
  
*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'  
  
  
*def* main():  
 contexts = get\_info\_from\_excel("../templates/09\_03\_01\_Информатика\_и\_ВТ,\_Матрица\_ВЕБ\_технологии\_2020.xlsx")  
 *for* key *in* contexts:  
 *try*:  
 context\_plane = get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx")[key]  
 *except KeyError*:  
 *for* error\_key *in* get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx"):  
 *if* SequenceMatcher(*None*, key, error\_key).ratio() >= 0.75:  
 context\_plane = get\_info\_from\_education\_plane("../media/excel/planes/03-5190 - ВЕБ 2020 (1).xlsx")[error\_key]  
 *break* 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* 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'])  
 doc = DocxTemplate("../templates/template.docx")  
 doc.render(contexts[key])  
 *for* i *in range*(*len*(doc.tables)):  
 table = doc.tables[i].\_tbl  
 *for* row *in* doc.tables[i].rows:  
 *if len*(row.cells[0].text.strip()) == 0 *and len*(*set*(row.cells)) == 1:  
 table.remove(row.\_tr)  
 doc.save("../generated\_files/{}.docx".format(key))  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 main()

**parser.py**

*import* datetime  
  
*import* openpyxl  
*import* xlrd  
  
  
*def* get\_parents(matrix, r):  
 scd = matrix[r][1].replace('\n', '')  
 fst = matrix[r][0].replace('\n', '')  
  
 *return* [fst, scd, matrix[r][2]]  
  
  
*def* get\_info\_for\_table(matrix, rng, c):  
 res = [  
 {  
 'competency\_code': '',  
 'competency\_name': '',  
 'indicators': [['', *set*()]]  
 }  
 ]  
  
 *for* r *in* rng:  
 *if* matrix[r][c] == '+':  
 f\_code, s\_code, t\_code = get\_parents(matrix, r)  
 *if* f\_code == '' *or* s\_code == '' *or* t\_code == '':  
 *continue* code, name = [el.strip() *for* el *in list*(*filter*(*bool*, f\_code.split('.')))]  
 *if* res[-1]['competency\_code'] != code:  
 res.append({  
 'competency\_code': code,  
 'competency\_name': name,  
 'indicators': [['', *set*()]]  
 })  
 res[-1]['indicators'][0][0] = s\_code  
 res[-1]['indicators'][0][1].add(t\_code)  
 *else*:  
 *if* res[-1]['indicators'][-1][0] != s\_code:  
 res[-1]['indicators'].append(['', *set*()])  
 res[-1]['indicators'][-1][0] = s\_code  
 res[-1]['indicators'][-1][1].add(t\_code)  
 *else*:  
 res[-1]['indicators'][-1][1].add(t\_code)  
 *del* res[0]  
 *return* res  
  
  
*def* get\_ranges(matrix):  
 rows = *len*(matrix)  
 skill\_types = []  
 k = 0  
 *for* i *in range*(rows)[1::]:  
 *if* matrix[i][2] == matrix[i][1] == matrix[i][0] *and* matrix[i][0] != '':  
 skill\_types += [*range*(k, i)]  
 k = i  
 *del* skill\_types[0]  
 skill\_types += [*range*(k, rows)]  
 *return* skill\_types  
  
  
*def* parse\_title(txt):  
 *import* re  
 res = {}  
 txt = re.sub('[»«]', '"', txt)  
 txt = re.sub('[\n,]', '', txt)  
 mas = txt.split('"')  
 res['profile\_name'] = mas[5]  
 res['program\_code'] = mas[3]  
 txt = re.sub('["]', ' ', txt)  
 mas = txt.split()  
 *for* el *in* mas:  
 *if* el.count('.') == 2:  
 res['program\_code'] = el + ' ' + res['program\_code']  
 *if* el.count('/') == 1:  
 temp = el.split('/')  
 res['year\_start'] = temp[0]  
 res['year\_end'] = temp[1]  
 *return* res  
  
  
*def* get\_matrix(filename):  
 xls = xlrd.open\_workbook(filename)  
 xls = xls.sheet\_by\_index(0)  
 mx\_row, mx\_column = xls.nrows, xls.ncols  
 wb = openpyxl.load\_workbook(filename)  
 sheet = wb.get\_sheet\_by\_name(wb.get\_sheet\_names()[0])  
 all\_data = []  
 *for* row\_index *in range*(1, mx\_row + 1):  
 row = []  
 *for* col\_index *in range*(1, mx\_column + 1):  
 vals = sheet.cell(row\_index, col\_index).value  
 *if* vals *is None*:  
 *for* crange *in* sheet.merged\_cells:  
 clo, rlo, chi, rhi = crange.bounds  
 top\_value = sheet.cell(rlo, clo).value  
 *if* rlo <= row\_index <= rhi *and* clo <= col\_index <= chi:  
 vals = top\_value  
 *break* row.append(vals)  
 *if len*(*list*(*filter*(*bool*, row))) > 0:  
 all\_data.append(row)  
 *for* i *in range*(*len*(all\_data)):  
 *for* j *in range*(*len*(all\_data[0])):  
 *if* all\_data[i][j] *is None*:  
 all\_data[i][j] = ''  
 all\_data[i][j] = *str*(all\_data[i][j]).strip()  
 *return* all\_data  
  
  
*def* get\_info\_from\_excel(filename):  
 matrix = get\_matrix(filename)  
 skill\_types = get\_ranges(matrix)  
 *for* i *in range*(*len*(matrix))[::-1]:  
 *if len*(*list*(*filter*(*bool*, matrix[i]))) == 0:  
 *del* matrix[i]  
 cols = *len*(matrix[0])  
 title = parse\_title(matrix[0][0])  
 data = {}  
 *for* c *in range*(cols)[3::]:  
 key = matrix[2][c]  
 *if* key == '':  
 *continue* data[key] = {}  
 data[key]['program\_name'] = key  
 data[key]['profile\_name'] = title['profile\_name']  
 data[key]['program\_code'] = title['program\_code']  
 data[key]['year\_start'] = title['year\_start']  
 data[key]['year\_end'] = title['year\_end']  
 data[key]['current\_year'] = *str*(datetime.date.today().year)  
 data[key]['part\_type'] = *str*.lower(matrix[1][c])  
 universal\_competences = get\_info\_for\_table(matrix, skill\_types[0], c)  
 general\_professional\_competencies = get\_info\_for\_table(matrix, skill\_types[1], c)  
 professional\_competencies = get\_info\_for\_table(matrix, skill\_types[2], c)  
 *if len*(universal\_competences) > 0:  
 data[key]['universal\_competences'] = universal\_competences  
 *if len*(general\_professional\_competencies) > 0:  
 data[key]['general\_professional\_competencies'] = general\_professional\_competencies  
 *if len*(professional\_competencies) > 0:  
 data[key]['professional\_competencies'] = professional\_competencies  
 *return* data

**parser\_plane.py**

*import* xlrd  
  
  
*def* get\_matrix(filename):  
 xls = xlrd.open\_workbook(filename)  
 xls = xls.sheet\_by\_index(0)  
  
 *return* [  
 [*str*(xls.cell\_value(i, j)).strip() *for* j *in range*(xls.ncols)]  
 *for* i *in range*(xls.nrows)  
 ]  
  
  
*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)  
  
  
*def* hours\_to\_zet(z):  
 h = *round*(z / 36, 1)  
 *if* h == *int*(h):  
 *return int*(h)  
 *else*:  
 *return* h  
  
  
*def* get\_courses(arr, imp\_cols, met='moduls'):  
 *if* met == 'practice':  
 sems = [  
 to\_int(el) *for* el *in* arr[imp\_cols['exam']].replace(' ', '').split(',')  
 ] + [  
 to\_int(el) *for* el *in* arr[imp\_cols['credit']].replace(' ', '').split(',')  
 ]  
 sems = *list*(*filter*(*lambda* x: x != 0, sems))  
 *return* [{  
 'semester': number\_to\_words(sem),  
 'course': number\_to\_words(*int*(*round*(sem / 2 + 0.1))),  
 'test': 'экзамен' *if str*(sem) *in* arr[imp\_cols['exam']] *else* 'зачет',  
 'hours': to\_int(arr[imp\_cols['ZET']]) \* 36,  
 'ZET': to\_int(arr[imp\_cols['ZET']]),  
 'homework\_time': 0,  
 } *for* sem *in* sems]  
  
 *if* met == 'elective':  
 sem = to\_int(arr[imp\_cols['elective\_sem']])  
 hours = to\_int(arr[imp\_cols['elective\_hours']])  
 *return* [{  
 'semester': number\_to\_words(sem),  
 'course': number\_to\_words(*int*(*round*(sem / 2 + 0.1))),  
 'test': 'зачет',  
 'hours': hours,  
 'ZET': hours\_to\_zet(hours),  
 'homework\_time': 0,  
 }]  
 courses = *list*(*map*(*lambda* el: [el[0] + 1, el[1]], *enumerate*(arr[imp\_cols['sems']::])))  
 courses = *list*(*filter*(*lambda* el: el[1] != '', courses))  
 courses = *list*(*map*(*lambda* el: [el[0], *float*(el[1])], courses))  
 courses\_count = *len*(courses)  
 all\_homework = to\_int(arr[imp\_cols['homework']])  
 homeworks = [[sem, time / 2] *for* sem, time *in* courses]  
 *if sum*([el[1] *for* el *in* homeworks]) != all\_homework:  
 homeworks = [[sem, time] *for* sem, time *in* courses]  
 *if sum*([el[1] *for* el *in* homeworks]) != all\_homework:  
 div = all\_homework // courses\_count  
 ost = all\_homework % courses\_count  
 homeworks = [[sem, div] *for* sem, \_ *in* courses]  
 idx = [el[1] *for* el *in* courses].index(*max*([el[1] *for* el *in* courses]))  
 homeworks[idx][1] += ost  
 homeworks = *dict*(homeworks)  
 res = []  
 *for* sem, time *in* courses:  
 res += [{}]  
 res[-1]['semester'] = number\_to\_words(sem)  
 res[-1]['course'] = number\_to\_words(*int*(*round*(sem / 2 + 0.1)))  
 res[-1]['test'] = 'экзамен' *if str*(sem) *in* arr[imp\_cols['exam']] *else* 'зачет'  
 res[-1]['homework\_time'] = to\_int(homeworks[sem])  
 res[-1]['hours'] = to\_int(time) + res[-1]['homework\_time']  
 res[-1]['ZET'] = hours\_to\_zet(to\_int(res[-1]['hours']))  
 *return* res  
  
  
*def* to\_int(x):  
 *try*:  
 *return int*(*float*(x))  
 *except Exception as* exc:  
 *print*(exc)  
 *return* 0  
  
  
*def* find\_from\_matrix(dct, matrix, idx=0):  
 rev = *dict*([[val, key] *for* key, val *in* dct.items()])  
 res = {}  
 *for* i *in range*(*len*(matrix)):  
 *for* j *in range*(*len*(matrix[0])):  
 *if* matrix[i][j] *in* dct.values() *and not* rev[matrix[i][j]] *in* res.keys():  
 res[rev[matrix[i][j]]] = i *if* idx == 0 *else* j  
 *return* res  
  
  
*def* get\_info\_from\_education\_plane(filename):  
 matrix = get\_matrix(filename)  
 imp\_rows = find\_from\_matrix({  
 'subjects': 'Обязательная часть',  
 'practice': 'Б.2',  
 'elective': 'Факультативные дисциплины',  
 }, matrix, 0)  
 imp\_cols = find\_from\_matrix({  
 'credit': 'зачетов',  
 'exam': 'экзаменов',  
 'hours': 'ВСЕГО по структуре',  
 'ZET': 'Всего, ЗЕТ',  
 'homework': 'Самостоятельная работа',  
 'sems': 'Распределение по курсам и семестрам, ауд. час.',  
 'subjects': 'Обязательная часть',  
 'B.1': 'Б.1',  
 'elective': 'Факультативные дисциплины',  
 'elective\_sem': 'Семестр',  
 'elective\_hours': 'Ауд. часов',  
 }, matrix, 1)  
 *for* i *in range*(*len*(matrix))[imp\_rows['subjects']::]:  
 matrix[i][imp\_cols['subjects']] = matrix[i][imp\_cols['subjects']].split('\*')[0].strip()  
 data = {}  
 *for* i *in range*(imp\_rows['subjects'], imp\_rows['practice']):  
 *if* matrix[i][imp\_cols['hours']] != '' *and* matrix[i][imp\_cols['B.1']] == '':  
 key = matrix[i][imp\_cols['subjects']]  
 data[key] = {}  
 data[key]['intensity\_hours'] = to\_int(matrix[i][imp\_cols['hours']])  
 data[key]['intensity\_ZET'] = hours\_to\_zet(to\_int(data[key]['intensity\_hours']))  
 data[key]['total\_homework\_hours'] = to\_int(matrix[i][imp\_cols['homework']])  
 data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='moduls')  
 *for* i *in range*(imp\_rows['practice'], imp\_rows['elective']):  
 *if* matrix[i][imp\_cols['B.1']] == '':  
 key = matrix[i][imp\_cols['subjects']]  
 data[key] = {}  
 data[key]['intensity\_ZET'] = to\_int(matrix[i][imp\_cols['ZET']])  
 data[key]['intensity\_hours'] = data[key]['intensity\_ZET'] \* 36  
 data[key]['total\_homework\_hours'] = 0  
 data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='practice')  
 i = imp\_rows['elective'] + 1  
 *while* matrix[i][imp\_cols['elective']] != '':  
 key = matrix[i][imp\_cols['elective']]  
 data[key] = {}  
 data[key]['intensity\_hours'] = to\_int(matrix[i][imp\_cols['elective\_hours']])  
 data[key]['intensity\_ZET'] = hours\_to\_zet(data[key]['intensity\_hours'])  
 data[key]['total\_homework\_hours'] = 0  
 data[key]['courses'] = get\_courses(matrix[i], imp\_cols, met='elective')  
 i += 1  
 *return* data

**views.py**

*import os.path  
import os.path  
import shutil  
from difflib import SequenceMatcher  
from os.path import join  
  
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 excel\_to\_doc\_parser.models import CustomUser, Role, Document, Theme, Section, Module, WorkProgram, ProgramNames, \  
 TimePlan  
from excel\_to\_doc\_parser.py.parser import get\_info\_from\_excel  
from excel\_to\_doc\_parser.py.parser\_plane import get\_info\_from\_education\_plane  
from parser\_server.settings import BASE\_DIR, MEDIA\_ROOT  
  
  
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()  
 program = fs.save(request.FILES['work\_program'].name, request.FILES['work\_program'])  
 time = fs.save(request.FILES['time\_plane'].name, request.FILES['time\_plane'])  
 work\_program, key\_list = get\_info\_from\_excel(join(MEDIA\_ROOT, program))  
 new\_program, created = WorkProgram.objects.update\_or\_create(profile\_name=work\_program['profile\_name'],  
 program\_code=work\_program["program\_code"],  
 year\_start=work\_program["year\_start"],  
 year\_end=work\_program["year\_end"])  
 for key in key\_list:  
 try:  
 time\_plane = get\_info\_from\_education\_plane(join(MEDIA\_ROOT, time))[key]  
 except KeyError:  
 for error\_key in get\_info\_from\_education\_plane(join(MEDIA\_ROOT, time)):  
 if SequenceMatcher(None, key, error\_key).ratio() >= 0.75:  
 time\_plane = get\_info\_from\_education\_plane(join(MEDIA\_ROOT, time))[  
 error\_key]  
 break  
 new\_program\_name, created = ProgramNames.objects.update\_or\_create(work\_program=new\_program,  
 program\_name=key)  
 new\_time\_plan, created = TimePlan.objects.update\_or\_create(program\_name=new\_program\_name,  
 classwork\_hours=time\_plane[  
 "intensity\_hours"],  
 homework\_hours=time\_plane[  
 "total\_homework\_hours"],  
 intensity\_ZET=time\_plane[  
 "intensity\_ZET"])  
 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)  
  
  
@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": ProgramNames.objects.all()}  
 if request.method == "POST":  
 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\_id=link, status\_id=status, user\_id=user,  
 program\_name=ProgramNames.objects.get(pk=ProgramNames.objects.get(program\_name=program\_name).id))  
 new\_document.save()  
 new\_theme = Theme(document\_id=new\_document)  
 new\_theme.save()  
 return redirect('/documents')  
 return render(request, "document.html", context)  
  
  
@login\_required(login\_url='/login/')  
def themes(request):  
 context = {}  
 theme = Theme.objects.get(document\_id=Document.objects.get(pk=request.GET.get("document")))  
 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["theme"] = Theme.objects.get(pk=theme.id)  
 context["modules"] = Module.objects.filter(theme\_id=Theme.objects.get(pk=theme.id))  
 context["homework\_hours"] = TimePlan.objects.get(program\_name=Document.objects.get(pk=request.GET.get("document")).program\_name).homework\_hours  
 context["classwork\_hours"] = TimePlan.objects.get(program\_name=Document.objects.get(pk=request.GET.get("document")).program\_name).classwork\_hours - context["homework\_hours"]  
 if len(context["modules"]) > 0:  
 context["last\_module"] = context["modules"].order\_by('-id')[0].module  
 context["sections"] = Section.objects.filter(theme\_id=context["theme"])  
 else:  
 context["last\_module"] = 0  
 if request.method == "POST":  
 if request.POST.get("generate"):  
 path = join(str(BASE\_DIR), "excel\_to\_doc\_parser/media/excel")  
 folder = join(str(BASE\_DIR), "excel\_to\_doc\_parser/media/generated\_files")  
 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, \_ = get\_info\_from\_excel(  
 path + "/matrices/" + "09\_03\_03\_Прикладная\_информатика,"  
 "\_Матрица\_Корпоративные\_информационные\_системы\_2020.xlsx")  
 discipline = "Навыки эффективной презентации"  
 print(data[discipline])  
 try:  
 context\_plane = get\_info\_from\_education\_plane(path + "/planes/03-5190 - ВЕБ 2020 (1).xlsx")[  
 discipline]  
 except KeyError:  
 for error\_key in get\_info\_from\_education\_plane(path + "/planes/planes/03-5190 - ВЕБ 2020 ("  
 "1).xlsx"):  
 if SequenceMatcher(None, discipline, error\_key).ratio() >= 0.75:  
 context\_plane = \  
 get\_info\_from\_education\_plane(path + "/planes/planes/03-5190 -"  
 " ВЕБ 2020 (1).xlsx")[error\_key]  
 break  
 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 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'])  
 context\_plane["modules"] = Module.objects.filter(theme\_id=Theme.objects.get(pk=theme.id))  
 if len(context["modules"]) > 0:  
 context\_plane["sections"] = Section.objects.filter(theme\_id=context["theme"])  
 doc = DocxTemplate(  
 join(str(BASE\_DIR), "excel\_to\_doc\_parser/templates/template.docx"))  
 doc.render(dict(data[discipline], \*\*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[0].text.strip()) == 0 and len(set(row.cells)) == 1:  
 table.remove(row.\_tr)  
 doc.save(join(str(BASE\_DIR), "excel\_to\_doc\_parser/media/generated\_files/{}.docx".format(  
 data[discipline]['program\_name'])))  
 context['path'] = "excel\_to\_doc\_parser/media/generated\_files/{}.docx".format(  
 data[discipline]['program\_name'])  
 context['name'] = data[discipline]['program\_name'] + '.docx'  
 return redirect("/download/?file={}&name=".format(context['path'], context["name"]))  
 if request.POST.get("new\_section"):  
 header = request.POST.get('new\_header')  
 description = request.POST.get('new\_description')  
 classwork\_hours = request.POST.get('new\_classwork')  
 homework\_hours = request.POST.get('new\_homework')  
 semester = request.POST.get('new\_semester')  
 week = request.POST.get('new\_week')  
 module = request.POST.get('new\_module')  
 theme = request.POST.get("new\_theme")  
 new\_module = Section(module\_id=Module.objects.get(pk=module), theme\_id=Theme.objects.get(pk=theme),  
 header=header, description=description,  
 classwork\_hours=classwork\_hours, homework\_hours=homework\_hours, semester=semester,  
 week=week)  
 new\_module.save()  
 return redirect("/themes/?document={}".format(request.GET.get("document")))  
 elif request.POST.get("new\_module"):  
 print(request.POST.get("theme"))  
 new\_module = Module(module=int(request.POST.get("last\_module")) + 1,  
 theme\_id\_id=request.POST.get("theme"))  
 new\_module.save()  
 return redirect("/themes/?document={}".format(request.GET.get("document")))  
 else:  
 pk = request.POST.get('pk')  
 header = request.POST.get('header')  
 description = request.POST.get('description')  
 classwork\_hours = request.POST.get('classwork')  
 homework\_hours = request.POST.get('homework')  
 semester = request.POST.get('semester')  
 week = request.POST.get('week')  
 module = Section.objects.filter(pk=pk)  
 module.update(header=header, description=description, classwork\_hours=classwork\_hours,  
 homework\_hours=homework\_hours, semester=semester, week=week)  
 return redirect("/themes/?document={}".format(request.GET.get("document")))  
 return render(request, "theme.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("/")  
 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("/")  
 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)*

**models.py**

*import datetime  
  
from django.contrib.auth.models import User  
from django.db import models  
  
  
class Role(models.Model):  
 role\_type = models.CharField(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 WorkProgram(models.Model):  
 profile\_name = models.CharField(max\_length=128)  
 program\_code = models.CharField(max\_length=128)  
 year\_start = models.IntegerField(default=datetime.date.today().year)  
 year\_end = models.IntegerField(default=datetime.date.today().year)  
  
  
class ProgramNames(models.Model):  
 work\_program = models.ForeignKey(WorkProgram, on\_delete=models.CASCADE)  
 program\_name = models.CharField(max\_length=512)  
  
  
class TimePlan(models.Model):  
 program\_name = models.ForeignKey(ProgramNames, on\_delete=models.CASCADE)  
 classwork\_hours = models.IntegerField(default=2)  
 homework\_hours = models.IntegerField(default=2)  
 intensity\_ZET = models.IntegerField(default=2)  
  
  
class Document(models.Model):  
 program\_name = models.ForeignKey(ProgramNames, on\_delete=models.CASCADE)  
 user = models.ForeignKey(CustomUser, on\_delete=models.CASCADE)  
 status = models.ForeignKey(Status, on\_delete=models.CASCADE)  
 link = models.ForeignKey(Link, on\_delete=models.CASCADE)  
  
  
class Theme(models.Model):  
 document\_id = models.ForeignKey(Document, on\_delete=models.CASCADE)  
  
  
class Module(models.Model):  
 module = models.IntegerField(default=1)  
 theme\_id = models.ForeignKey(Theme, on\_delete=models.CASCADE)  
  
  
class Section(models.Model):  
 header = models.CharField(max\_length=128)  
 classwork\_hours = models.IntegerField(default=2)  
 homework\_hours = models.IntegerField(default=2)  
 description = models.TextField()  
 module\_id = models.ForeignKey(Module, on\_delete=models.CASCADE)  
 theme\_id = models.ForeignKey(Theme, on\_delete=models.CASCADE)  
 semester = models.IntegerField(default=1)  
 week = models.IntegerField(default=1)*

**settings.py**

*"""  
Django settings for parser\_server project.  
  
Generated by 'django-admin startproject' using Django 4.0.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/4.0/topics/settings/  
  
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/4.0/ref/settings/  
"""  
import* os.path  
*from* os.path *import* join  
*from* pathlib *import* Path  
  
*# Build paths inside the project like this: BASE\_DIR / 'subdir'.*BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
  
  
*# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/4.0/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!*SECRET\_KEY = 'django-insecure-j!uaz5%mg^i7h&nac)59pzz@lq)l&g10#\_+ugwn)bt9qcjp%g#'  
  
*# SECURITY WARNING: don't run with debug turned on in production!*DEBUG = *True*ALLOWED\_HOSTS = []  
  
  
*# Application definition*INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 'django.contrib.contenttypes',  
 'django.contrib.sessions',  
 'django.contrib.messages',  
 'django.contrib.staticfiles',  
 'excel\_to\_doc\_parser'  
]  
  
MIDDLEWARE = [  
 'django.middleware.security.SecurityMiddleware',  
 'django.contrib.sessions.middleware.SessionMiddleware',  
 'django.middleware.common.CommonMiddleware',  
 'django.middleware.csrf.CsrfViewMiddleware',  
 'django.contrib.auth.middleware.AuthenticationMiddleware',  
 'django.contrib.messages.middleware.MessageMiddleware',  
 'django.middleware.clickjacking.XFrameOptionsMiddleware',  
]  
  
ROOT\_URLCONF = 'parser\_server.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [join(BASE\_DIR, 'excel\_to\_doc\_parser/templates/html')],  
 'APP\_DIRS': *True*,  
 'OPTIONS': {  
 'context\_processors': [  
 'django.template.context\_processors.debug',  
 'django.template.context\_processors.request',  
 'django.contrib.auth.context\_processors.auth',  
 'django.contrib.messages.context\_processors.messages',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'parser\_server.wsgi.application'  
  
  
*# Database  
# https://docs.djangoproject.com/en/4.0/ref/settings/#databases*DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': os.environ['DB\_NAME'],  
 'USER': os.environ['DB\_USER'],  
 'PASSWORD': os.environ['DB\_PASSWORD'],  
 'HOST': os.environ['DB\_HOST'],  
 'PORT': os.environ['DB\_PORT']  
 }  
}  
  
  
*# Password validation  
# https://docs.djangoproject.com/en/4.0/ref/settings/#auth-password-validators*AUTH\_PASSWORD\_VALIDATORS = [  
 {  
 'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',  
 },  
 {  
 'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',  
 },  
]  
  
  
*# Internationalization  
# https://docs.djangoproject.com/en/4.0/topics/i18n/*LANGUAGE\_CODE = 'ru-ru'  
  
TIME\_ZONE = 'Europe/Moscow'  
  
USE\_I18N = *True*USE\_TZ = *True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/4.0/howto/static-files/*MEDIA\_DIR = join(BASE\_DIR, 'media')  
STATIC\_URL = '/static/'  
MEDIA\_ROOT = MEDIA\_DIR  
MEDIA\_URL = '/media/'  
STATIC\_ROOT = join(BASE\_DIR, 'excel\_to\_doc\_parser/staticfiles/')  
  
*# Default primary key field type  
# https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field*DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'  
LOGIN\_URL = '/login/'  
LOGIN\_REDIRECT\_URL = '/'  
LOGOUT\_LOGIN\_REDIRECT = '/login/'