Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**«Московский государственный технический университет имени Н.Э. Баумана»   
(МГТУ им. Н.Э. Баумана)**

**Отчет по домашнему заданию**

**по дисциплине «Разработка интернет приложений».**

|  |
| --- |
| ИСПОЛНИТЕЛЬ: |
| студент группы ИУ5-54 |
| Василенко Н.А. |

Москва, 2017

## Задание

Разработать веб­сервис на базе технологий: Python, Django, JS, MySQL.

7  Пользователь  Членство  Кафедра

Код программы

My\_style.css

.razm{  
 max-width: 50%;  
 margin-left: auto;  
 margin-right: auto;  
}  
.form-signin{  
 max-width: 60%;  
 padding: 15px;  
 margin: 0 auto;  
}  
.error{  
 color: red;  
}  
.good{  
 color: green;  
}  
  
.img-inform{  
 max-width: 100%;  
 max-height: 100%;  
}  
  
.img-icon{  
 max-width: 40px;  
 max-height: 40px;  
 margin-top: -10px;  
}  
  
.color-nav {  
  
}  
  
.color-text{  
 color: black **!important**;  
}  
  
body{  
 background-color: #ffffff **!important**;  
}  
  
.img-size{  
 max-width: 100px;  
 max-height: 100px;  
}  
  
.size-osn{  
 max-width: 70%;  
 margin-right: auto;  
 margin-left: auto;  
}  
  
.osn-color{  
 background-color: #FFFFFF;  
}  
.color-ss{  
 color:black **!important**;  
 margin-left:auto;  
 margin-right:auto;  
 text-align: center;  
}  
.pag-color{  
 color:black **!important**;  
 background-color: white **!important**;  
}  
  
.m-inf{  
 margin-bottom: 0px **!important**;  
}

Admin.py

from django.contrib import admin  
  
# Register your models here.  
from django.contrib import admin  
from django.contrib import admin  
from dz2.models import group  
from dz2.models import Group\_user  
admin.site.register(group)  
admin.site.register(Group\_user)  
# Register your models here.

Form.py

from \_\_future\_\_ import unicode\_literals  
  
import unicodedata  
  
  
from django.core.wsgi import get\_wsgi\_application  
  
application = get\_wsgi\_application()  
from django import forms  
from django.contrib.auth import (  
 authenticate, get\_user\_model, password\_validation,  
)  
from django.contrib.auth.hashers import (  
 UNUSABLE\_PASSWORD\_PREFIX, identify\_hasher,  
)  
from django.contrib.auth.models import User  
from django.contrib.auth.tokens import default\_token\_generator  
from django.contrib.sites.shortcuts import get\_current\_site  
from django.core.mail import EmailMultiAlternatives  
from django.template import loader  
from django.utils.encoding import force\_bytes  
from django.utils.http import urlsafe\_base64\_encode  
from django.utils.text import capfirst  
from django.utils.translation import ugettext, ugettext\_lazy as \_  
from dz2 import models  
  
UserModel = get\_user\_model()  
  
  
class ReadOnlyPasswordHashWidget(forms.Widget):  
 template\_name = 'auth/widgets/read\_only\_password\_hash.html'  
  
 def get\_context(self, name, value, attrs):  
 context = super(ReadOnlyPasswordHashWidget, self).get\_context(name, value, attrs)  
 summary = []  
 if not value or value.startswith(UNUSABLE\_PASSWORD\_PREFIX):  
 summary.append({'label': ugettext("No password set.")})  
 else:  
 try:  
 hasher = identify\_hasher(value)  
 except ValueError:  
 summary.append({'label': ugettext("Invalid password format or unknown hashing algorithm.")})  
 else:  
 for key, value\_ in hasher.safe\_summary(value).items():  
 summary.append({'label': ugettext(key), 'value': value\_})  
 context['summary'] = summary  
 return context  
  
  
class ReadOnlyPasswordHashField(forms.Field):  
 widget = ReadOnlyPasswordHashWidget  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 kwargs.setdefault("required", False)  
 super(ReadOnlyPasswordHashField, self).\_\_init\_\_(\*args, \*\*kwargs)  
  
 def bound\_data(self, data, initial):  
 # Always return initial because the widget doesn't  
 # render an input field.  
 return initial  
  
 def has\_changed(self, initial, data):  
 return False  
  
  
class UsernameField(forms.CharField):  
 def to\_python(self, value):  
 return unicodedata.normalize('NFKC', super(UsernameField, self).to\_python(value))  
  
  
class UserCreationForm(forms.ModelForm):  
 *"""  
 A form that creates a user, with no privileges, from the given username and  
 password.  
 """* error\_messages = {  
 'password\_mismatch': \_("The two password fields didn't match."),  
 }  
 password1 = forms.CharField(  
 label=\_("Password"),  
 strip=False,  
 widget=forms.PasswordInput,  
 help\_text=password\_validation.password\_validators\_help\_text\_html(),  
 )  
 password2 = forms.CharField(  
 label=\_("Password confirmation"),  
 widget=forms.PasswordInput,  
 strip=False,  
 help\_text=\_("Enter the same password as before, for verification."),  
 )  
  
 class Meta:  
 model = User  
 fields = ("username", "email", 'last\_name', 'first\_name')  
 field\_classes = {'username': UsernameField}  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super(UserCreationForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
 if self.\_meta.model.USERNAME\_FIELD in self.fields:  
 self.fields[self.\_meta.model.USERNAME\_FIELD].widget.attrs.update({'autofocus': True})  
 self.fields['username'].widget.attrs.update({'class': 'form-control'})  
 self.fields['password1'].widget.attrs.update({'class': 'form-control'})  
 self.fields['password2'].widget.attrs.update({'class': 'form-control'})  
 self.fields['email'].widget.attrs.update({'class': 'form-control'})  
 self.fields['email'].widget.attrs.update({'id': 'inputEmail'})  
 self.fields['last\_name'].widget.attrs.update({'class': 'form-control'})  
 self.fields['first\_name'].widget.attrs.update({'class': 'form-control'})  
  
 def clean\_password2(self):  
 password1 = self.cleaned\_data.get("password1")  
 password2 = self.cleaned\_data.get("password2")  
 if password1 and password2 and password1 != password2:  
 raise forms.ValidationError(  
 self.error\_messages['password\_mismatch'],  
 code='password\_mismatch',  
 )  
 self.instance.username = self.cleaned\_data.get('username')  
 password\_validation.validate\_password(self.cleaned\_data.get('password2'), self.instance)  
 return password2  
  
 def save(self, commit=True):  
 user = super(UserCreationForm, self).save(commit=False)  
 user.set\_password(self.cleaned\_data["password1"])  
 if commit:  
 user.save()  
 return user  
  
  
class UserChangeForm(forms.ModelForm):  
 password = ReadOnlyPasswordHashField(  
 label=\_("Password"),  
 help\_text=\_(  
 "Raw passwords are not stored, so there is no way to see this "  
 "user's password, but you can change the password using "  
 "<a href=\"../password/\">this form</a>."  
 ),  
 )  
  
 class Meta:  
 model = User  
 fields = '\_\_all\_\_'  
 field\_classes = {'username': UsernameField}  
  
 def \_\_init\_\_(self, \*args, \*\*kwargs):  
 super(UserChangeForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
 f = self.fields.get('user\_permissions')  
 if f is not None:  
 f.queryset = f.queryset.select\_related('content\_type')  
  
 def clean\_password(self):  
 # Regardless of what the user provides, return the initial value.  
 # This is done here, rather than on the field, because the  
 # field does not have access to the initial value  
 return self.initial["password"]  
  
  
class AuthenticationForm(forms.Form):  
 *"""  
 Base class for authenticating users. Extend this to get a form that accepts  
 username/password logins.  
 """* username = UsernameField(  
 max\_length=254,  
 widget=forms.TextInput(attrs={'autofocus': True}),  
 )  
 password = forms.CharField(  
 label=\_("Password"),  
 strip=False,  
 widget=forms.PasswordInput,  
 )  
  
 error\_messages = {  
 'invalid\_login': \_(  
 "Please enter a correct %(username)s and password. Note that both "  
 "fields may be case-sensitive."  
 ),  
 'inactive': \_("This account is inactive."),  
 }  
  
 def \_\_init\_\_(self, request=None, \*args, \*\*kwargs):  
 *"""  
 The 'request' parameter is set for custom auth use by subclasses.  
 The form data comes in via the standard 'data' kwarg.  
 """* self.request = request  
 self.user\_cache = None  
 super(AuthenticationForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
  
 # Set the label for the "username" field.  
 self.username\_field = UserModel.\_meta.get\_field(UserModel.USERNAME\_FIELD)  
 if self.fields['username'].label is None:  
 self.fields['username'].label = capfirst(self.username\_field.verbose\_name)  
 self.fields['username'].widget.attrs.update({'class': 'form-control'})  
 self.fields['password'].widget.attrs.update({'class': 'form-control'})  
  
 def clean(self):  
 username = self.cleaned\_data.get('username')  
 password = self.cleaned\_data.get('password')  
  
 if username is not None and password:  
 self.user\_cache = authenticate(self.request, username=username, password=password)  
 if self.user\_cache is None:  
 raise forms.ValidationError(  
 self.error\_messages['invalid\_login'],  
 code='invalid\_login',  
 params={'username': self.username\_field.verbose\_name},  
 )  
 else:  
 self.confirm\_login\_allowed(self.user\_cache)  
  
 return self.cleaned\_data  
  
 def confirm\_login\_allowed(self, user):  
 *"""  
 Controls whether the given User may log in. This is a policy setting,  
 independent of end-user authentication. This default behavior is to  
 allow login by active users, and reject login by inactive users.  
  
 If the given user cannot log in, this method should raise a  
 ``forms.ValidationError``.  
  
 If the given user may log in, this method should return None.  
 """* if not user.is\_active:  
 raise forms.ValidationError(  
 self.error\_messages['inactive'],  
 code='inactive',  
 )  
  
 def get\_user\_id(self):  
 if self.user\_cache:  
 return self.user\_cache.id  
 return None  
  
 def get\_user(self):  
 return self.user\_cache  
  
  
class PasswordResetForm(forms.Form):  
 email = forms.EmailField(label=\_("Email"), max\_length=254)  
  
 def send\_mail(self, subject\_template\_name, email\_template\_name,  
 context, from\_email, to\_email, html\_email\_template\_name=None):  
 *"""  
 Sends a django.core.mail.EmailMultiAlternatives to `to\_email`.  
 """* subject = loader.render\_to\_string(subject\_template\_name, context)  
 # Email subject \*must not\* contain newlines  
 subject = ''.join(subject.splitlines())  
 body = loader.render\_to\_string(email\_template\_name, context)  
  
 email\_message = EmailMultiAlternatives(subject, body, from\_email, [to\_email])  
 if html\_email\_template\_name is not None:  
 html\_email = loader.render\_to\_string(html\_email\_template\_name, context)  
 email\_message.attach\_alternative(html\_email, 'text/html')  
  
 email\_message.send()  
  
 def get\_users(self, email):  
 *"""Given an email, return matching user(s) who should receive a reset.  
  
 This allows subclasses to more easily customize the default policies  
 that prevent inactive users and users with unusable passwords from  
 resetting their password.  
 """* active\_users = UserModel.\_default\_manager.filter(\*\*{  
 '%s\_\_iexact' % UserModel.get\_email\_field\_name(): email,  
 'is\_active': True,  
 })  
 return (u for u in active\_users if u.has\_usable\_password())  
  
 def save(self, domain\_override=None,  
 subject\_template\_name='registration/password\_reset\_subject.txt',  
 email\_template\_name='registration/password\_reset\_email.html',  
 use\_https=False, token\_generator=default\_token\_generator,  
 from\_email=None, request=None, html\_email\_template\_name=None,  
 extra\_email\_context=None):  
 *"""  
 Generates a one-use only link for resetting password and sends to the  
 user.  
 """* email = self.cleaned\_data["email"]  
 for user in self.get\_users(email):  
 if not domain\_override:  
 current\_site = get\_current\_site(request)  
 site\_name = current\_site.name  
 domain = current\_site.domain  
 else:  
 site\_name = domain = domain\_override  
 context = {  
 'email': email,  
 'domain': domain,  
 'site\_name': site\_name,  
 'uid': urlsafe\_base64\_encode(force\_bytes(user.pk)),  
 'user': user,  
 'token': token\_generator.make\_token(user),  
 'protocol': 'https' if use\_https else 'http',  
 }  
 if extra\_email\_context is not None:  
 context.update(extra\_email\_context)  
 self.send\_mail(  
 subject\_template\_name, email\_template\_name, context, from\_email,  
 email, html\_email\_template\_name=html\_email\_template\_name,  
 )  
  
  
class SetPasswordForm(forms.Form):  
 *"""  
 A form that lets a user change set their password without entering the old  
 password  
 """* error\_messages = {  
 'password\_mismatch': \_("The two password fields didn't match."),  
 }  
 new\_password1 = forms.CharField(  
 label=\_("New password"),  
 widget=forms.PasswordInput,  
 strip=False,  
 help\_text=password\_validation.password\_validators\_help\_text\_html(),  
 )  
 new\_password2 = forms.CharField(  
 label=\_("New password confirmation"),  
 strip=False,  
 widget=forms.PasswordInput,  
 )  
  
 def \_\_init\_\_(self, user, \*args, \*\*kwargs):  
 self.user = user  
 super(SetPasswordForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
  
 def clean\_new\_password2(self):  
 password1 = self.cleaned\_data.get('new\_password1')  
 password2 = self.cleaned\_data.get('new\_password2')  
 if password1 and password2:  
 if password1 != password2:  
 raise forms.ValidationError(  
 self.error\_messages['password\_mismatch'],  
 code='password\_mismatch',  
 )  
 password\_validation.validate\_password(password2, self.user)  
 return password2  
  
 def save(self, commit=True):  
 password = self.cleaned\_data["new\_password1"]  
 self.user.set\_password(password)  
 if commit:  
 self.user.save()  
 return self.user  
  
  
class PasswordChangeForm(SetPasswordForm):  
 *"""  
 A form that lets a user change their password by entering their old  
 password.  
 """* error\_messages = dict(SetPasswordForm.error\_messages, \*\*{  
 'password\_incorrect': \_("Your old password was entered incorrectly. Please enter it again."),  
 })  
 old\_password = forms.CharField(  
 label=\_("Old password"),  
 strip=False,  
 widget=forms.PasswordInput(attrs={'autofocus': True}),  
 )  
  
 field\_order = ['old\_password', 'new\_password1', 'new\_password2']  
  
 def clean\_old\_password(self):  
 *"""  
 Validates that the old\_password field is correct.  
 """* old\_password = self.cleaned\_data["old\_password"]  
 if not self.user.check\_password(old\_password):  
 raise forms.ValidationError(  
 self.error\_messages['password\_incorrect'],  
 code='password\_incorrect',  
 )  
 return old\_password  
  
  
class AdminPasswordChangeForm(forms.Form):  
 *"""  
 A form used to change the password of a user in the admin interface.  
 """* error\_messages = {  
 'password\_mismatch': \_("The two password fields didn't match."),  
 }  
 required\_css\_class = 'required'  
 password1 = forms.CharField(  
 label=\_("Password"),  
 widget=forms.PasswordInput(attrs={'autofocus': True}),  
 strip=False,  
 help\_text=password\_validation.password\_validators\_help\_text\_html(),  
 )  
 password2 = forms.CharField(  
 label=\_("Password (again)"),  
 widget=forms.PasswordInput,  
 strip=False,  
 help\_text=\_("Enter the same password as before, for verification."),  
 )  
  
 def \_\_init\_\_(self, user, \*args, \*\*kwargs):  
 self.user = user  
 super(AdminPasswordChangeForm, self).\_\_init\_\_(\*args, \*\*kwargs)  
  
 def clean\_password2(self):  
 password1 = self.cleaned\_data.get('password1')  
 password2 = self.cleaned\_data.get('password2')  
 if password1 and password2:  
 if password1 != password2:  
 raise forms.ValidationError(  
 self.error\_messages['password\_mismatch'],  
 code='password\_mismatch',  
 )  
 password\_validation.validate\_password(password2, self.user)  
 return password2  
  
 def save(self, commit=True):  
 *"""  
 Saves the new password.  
 """* password = self.cleaned\_data["password1"]  
 self.user.set\_password(password)  
 if commit:  
 self.user.save()  
 return self.user  
  
 @property  
 def changed\_data(self):  
 data = super(AdminPasswordChangeForm, self).changed\_data  
 for name in self.fields.keys():  
 if name not in data:  
 return []  
 return ['password']  
  
class group\_form(forms.ModelForm):  
 picture = forms.FileField(  
 widget=forms.ClearableFileInput(),#attrs={'class': 'ask-signup-avatar-input', }),  
 required=False, label=u'Логотип'  
 )  
 name = forms.CharField()  
  
  
 description = forms.CharField()  
 class Meta:  
 model = models.group  
 fields = ['name', 'description', 'picture']  
  
 def \_\_init\_\_(self, request=None, \*args, \*\*kwargs):  
 self.request = request  
 super(group\_form, self).\_\_init\_\_(request, \*args, \*\*kwargs)  
  
 self.fields['name'].widget.attrs.update({'class': 'form-control'})  
  
 self.fields['description'].widget.attrs.update({'class': 'form-control'})  
 self.fields['picture'].widget.attrs.update({'class': 'btn btn-default'})#!!!!!!!!!!!!!!!!!!!!!!!!!

Models.py

import os  
from django.db import models  
from django.contrib.auth.models import User  
  
  
  
def avatar\_upload\_to(instance, filename):  
 return os.path.join('uploads', instance.name + os.path.splitext(filename)[1])  
  
class group(models.Model):  
 id = models.IntegerField(primary\_key=True)  
 name = models.CharField(max\_length=30)  
 description = models.CharField(max\_length=255)  
 picture = models.ImageField(upload\_to=avatar\_upload\_to)  
# Create your models here.  
  
class Group\_user(models.Model):  
 group\_id = models.ForeignKey(group, on\_delete=models.CASCADE)  
 user\_id = models.ForeignKey(User, on\_delete=models.CASCADE)

urls.py

from django.conf.urls import url  
from . import views  
urlpatterns = [  
url(r'^$', views.group, name='group'),  
url(r'^$', views.group\_user, name='group\_user')  
]

views.py

from django.shortcuts import render  
from django.views.generic.edit import FormView  
#from django.contrib.auth.forms import UserCreationForm  
# Опять же, спасибо django за готовую форму аутентификации.  
#from django.contrib.auth.forms import AuthenticationForm  
from dz2.form import UserCreationForm  
from dz2.form import AuthenticationForm  
# Функция для установки сессионного ключа.  
# По нему django будет определять, выполнил ли вход пользователь.  
from django.contrib.auth import login  
from django.http import HttpResponseRedirect  
from django.views.generic.base import View  
from django.views.generic.list import ListView  
from django.contrib.auth import logout  
from dz2 import models  
from django.contrib.auth.models import User  
from dz2 import form  
from django.shortcuts import redirect, reverse  
from django.core.paginator import Paginator, EmptyPage, PageNotAnInteger  
  
class LogoutView(View):  
 def get(self, request):  
 # Выполняем выход для пользователя, запросившего данное представление.  
 logout(request)  
  
 # После чего, перенаправляем пользователя на главную страницу.  
 return HttpResponseRedirect("/")  
  
class LoginFormView(FormView):  
 form\_class = AuthenticationForm  
  
 # Аналогично регистрации, только используем шаблон аутентификации.  
 template\_name = "login.html"  
  
 # В случае успеха перенаправим на главную.  
 success\_url = "/"  
  
 def form\_valid(self, form):  
 # Получаем объект пользователя на основе введённых в форму данных.  
 self.user = form.get\_user()  
  
 # Выполняем аутентификацию пользователя.  
 login(self.request, self.user)  
 return super(LoginFormView, self).form\_valid(form)  
  
class RegisterFormView(FormView):  
 form\_class = UserCreationForm  
  
 # Ссылка, на которую будет перенаправляться пользователь в случае успешной регистрации.  
 # В данном случае указана ссылка на страницу входа для зарегистрированных пользователей.  
 success\_url = "/login/"  
  
 # Шаблон, который будет использоваться при отображении представления.  
 template\_name = "register.html"  
  
 def form\_valid(self, form):  
 # Создаём пользователя, если данные в форму были введены корректно.  
 form.save()  
  
 # Вызываем метод базового класса  
 return super(RegisterFormView, self).form\_valid(form)  
# Create your views here.  
  
class groups\_view(ListView):  
 model = models.group  
  
 def get(self, request):  
 d = self.model.objects.values('id','name','picture')  
 paginator = Paginator(d, 3) # Show 25 contacts per page  
 page = request.GET.get('page')  
 try:  
 contacts = paginator.page(page)  
 except PageNotAnInteger:  
 # If page is not an integer, deliver first page.  
 contacts = paginator.page(1)  
 except EmptyPage:  
 # If page is out of range (e.g. 9999), deliver last page of results.  
 contacts = paginator.page(paginator.num\_pages)  
 if(contacts.paginator.num\_pages <= 4):  
 r = range(1, contacts.paginator.num\_pages+1)  
 else:  
 r = range(contacts.number-1, contacts.number+4)  
 return render(request, 'groups.html', {"contacts": contacts, 'user': request.user, 'range': r})  
  
class group\_view(View):  
 model = models.group  
 model1 = models.Group\_user  
 model2 = User  
 def get(self, request,id1):  
  
 b = dict(group=self.model.objects.filter(id=id1))  
  
 a = dict(users=self.model1.objects.filter(group\_id=id1))  
  
 b.update(a)  
 c={'user':request.user}  
 b.update(c)  
 d={'id':id1}  
 b.update(d)  
 return render(request, "inform\_group.html", b)  
  
class create\_group(FormView):  
 form\_class = form.group\_form  
 model = models.group  
 template\_name = "create\_group.html"  
 success\_url = "/groups/"  
 def form\_valid(self, form):  
 # Создаём пользователя, если данные в форму были введены корректно.  
 form.save()  
  
 # Вызываем метод базового класса  
 return super(create\_group, self).form\_valid(form)  
  
 def post(self, request, \*args, \*\*kwargs):  
 form1 = form.group\_form(request.POST)  
 if form1.is\_valid() and request.FILES.get('picture'):  
 obj = models.group()  
 obj = form1.save(commit=False)  
 obj.picture = request.FILES['picture']  
 obj.save()  
 b = dict(group=self.model.objects.filter(name=obj.name))  
 return HttpResponseRedirect("/group/"+str(b['group'][0].id))  
 else:  
 form1.add\_error('picture', 'Изображение')  
 return render(request, 'create\_group.html', {'form':form1})  
  
class add(View):  
 model = models.group  
 model1 = models.Group\_user  
 model2 = User  
 def get(self, request, id2):  
 if request.user.is\_authenticated:  
 u = self.model2.objects.filter(id=request.user.id)  
 a = dict(users=self.model1.objects.filter(group\_id=id2))  
 b = dict(group=self.model.objects.filter(id=id2))  
 ab = dict(users=self.model1.objects.filter(group\_id=id2))  
 b.update(ab)  
 er = 0  
 for i in a['users']:  
 if i.user\_id == request.user:  
 er = 1  
 break  
 if er == 1:  
 ca = {'uze': "Вы уже член этой кафедры"}  
 b.update(ca)  
 k = {'message':""}  
 ca = {'error':''}  
 b.update(k)  
 b.update(ca)  
 d = {'id': id2}  
 b.update(d)  
 return render(request, "inform\_group.html", b)  
 else:  
  
 a=models.Group\_user()  
  
 a.group\_id=b['group'][0]  
 a.user\_id=u[0]  
 a.save()  
  
 k = {'message':"Вы стали членом кафедры"}  
 ca = {'error':''}  
 b.update(k)  
 b.update(ca)  
 d = {'id': id2}  
 b.update(d)  
  
 return render(request, "inform\_group.html", b)  
 else:  
 b = dict(group=self.model.objects.filter(id=id2))  
 a = {'error': "Вы не авторизованны. Войдите или зарегистрируйтесь"}  
 c = {'message':''}  
 b.update(a)  
 b.update(c)  
 d = {'id': id2}  
 b.update(d)  
 ab = dict(users=self.model1.objects.filter(group\_id=id2))  
 b.update(ab)  
 return render(request, "inform\_group.html", b)

urls.py

*"""dz\_v2 URL Configuration  
  
The `urlpatterns` list routes URLs to views. For more information please see:  
 https://docs.djangoproject.com/en/1.11/topics/http/urls/  
Examples:  
Function views  
 1. Add an import: from my\_app import views  
 2. Add a URL to urlpatterns: url(r'^$', views.home, name='home')  
Class-based views  
 1. Add an import: from other\_app.views import Home  
 2. Add a URL to urlpatterns: url(r'^$', Home.as\_view(), name='home')  
Including another URLconf  
 1. Import the include() function: from django.conf.urls import url, include  
 2. Add a URL to urlpatterns: url(r'^blog/', include('blog.urls'))  
"""*from django.conf.urls import url  
from django.contrib import admin  
from dz2 import views  
from django.conf.urls.static import static  
  
from django.contrib.staticfiles.urls import staticfiles\_urlpatterns  
  
from django.conf import settings  
urlpatterns = [  
  
 url(r'^admin/', admin.site.urls),  
 url(r'^register/$', views.RegisterFormView.as\_view()),  
 url(r'^login/$', views.LoginFormView.as\_view()),  
 url(r'^logout/$', views.LogoutView.as\_view()),  
 url(r'^groups/$', views.groups\_view.as\_view()),  
 url(r'^group/(?P<id1>\d+)$', views.group\_view.as\_view(), name='group\_url'),  
 url(r'^create\_group/$', views.create\_group.as\_view()),  
 url(r'^add/(?P<id2>\d+)$', views.add.as\_view(), name='add\_url'),  
]+ static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)+ static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)  
  
urlpatterns.append(url(r'$', views.groups\_view.as\_view()))

Settings.py

*"""  
Django settings for dz\_v2 project.  
  
Generated by 'django-admin startproject' using Django 1.11.6.  
  
For more information on this file, see  
https://docs.djangoproject.com/en/1.11/topics/settings/  
  
For the full list of settings and their values, see  
https://docs.djangoproject.com/en/1.11/ref/settings/  
"""*import os  
  
# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
PROJECT\_DIR = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '..'))  
  
  
# Quick-start development settings - unsuitable for production  
# See https://docs.djangoproject.com/en/1.11/howto/deployment/checklist/  
  
# SECURITY WARNING: keep the secret key used in production secret!  
SECRET\_KEY = 'h^\_213+++tsq\_$q&w8pw=ndxem-g^b4nt69ud$lv!\_%g\*40y4q'  
  
# 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',  
 'dz2.apps.Dz2Config',  
]  
  
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 = 'dz\_v2.urls'  
  
TEMPLATES = [  
 {  
 'BACKEND': 'django.template.backends.django.DjangoTemplates',  
 'DIRS': [os.path.join(BASE\_DIR, 'templates')]  
 ,  
 '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',  
 'django.template.context\_processors.media',  
 ],  
 },  
 },  
]  
  
WSGI\_APPLICATION = 'dz\_v2.wsgi.application'  
  
  
# Database  
# https://docs.djangoproject.com/en/1.11/ref/settings/#databases  
  
DATABASES = {  
 'default': {  
 'ENGINE': 'django.db.backends.mysql',  
 'NAME': 'db',  
 'USER': 'dbuser1',  
 'PASSWORD': '1',  
 'HOST': 'localhost',  
 'PORT': 3306,  
 'OPTIONS': {'charset':'utf8'},  
 'TEST\_CHARSET': 'utf8',  
 }  
}  
  
  
# Password validation  
# https://docs.djangoproject.com/en/1.11/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/1.11/topics/i18n/  
  
LANGUAGE\_CODE = 'en-us'  
  
TIME\_ZONE = 'UTC'  
  
USE\_I18N = True  
  
USE\_L10N = True  
  
USE\_TZ = True  
  
  
# Static files (CSS, JavaScript, Images)  
# https://docs.djangoproject.com/en/1.11/howto/static-files/  
  
STATIC\_URL = '/static/'  
STATIC\_ROOT = os.path.join(PROJECT\_DIR, 'static')  
  
MEDIA\_ROOT = os.path.join(BASE\_DIR, 'uploads')  
MEDIA\_URL = '/uploads/'

Base.html

{% load static %}  
  
<!DOCTYPE html>  
<html lang="en">  
 <head>  
 <meta charset="utf-8">  
 <meta http-equiv="X-UA-Compatible" content="IE=edge">  
 <meta name="viewport" content="width=device-width, initial-scale=1">  
 <!-- The above 3 meta tags \*must\* come first in the head; any other head content must come \*after\* these tags -->  
 <title>{% block title %}{% endblock %}</title>  
 <!--<link href="css/my\_style.css" rel="stylesheet" type="text/css">-->  
  
 <!-- Bootstrap -->  
 <!--<link href="css/bootstrap.css" rel="stylesheet">-->  
 <link rel="stylesheet" type="text/css" href="{% static 'css/bootstrap.css' %}">  
 <link rel="stylesheet" type="text/css" href="{% static 'css/my\_style.css' %}">  
 <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->  
 <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->  
 <!--[if lt IE 9]>  
 <script src="https://oss.maxcdn.com/html5shiv/3.7.3/html5shiv.min.js"></script>  
 <script src="https://oss.maxcdn.com/respond/1.4.2/respond.min.js"></script>  
 <![endif]-->  
 </head>  
 <body>  
 <div class="razm">  
  
 {% block body %}Переопределяется в наследниках{% endblock %}  
 </div>  
  
  
  
 <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->  
 <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>  
 <!-- Include all compiled plugins (below), or include individual files as needed -->  
 <script src="{% static 'js/bootstrap.min.js' %}"></script>  
 </body>  
</html>

groups.html

{% extends 'base.html' %}  
{% load static %}  
{% block title %}Кафедры{% endblock %}  
{% block body %}  
 <nav class="navbar navbar-default color-nav">  
 <div class="container-fluid">  
 <!-- Brand and toggle get grouped for better mobile display -->  
 <div class="navbar-header">  
 <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">  
 <span class="sr-only">Toggle navigation</span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 </button>  
 <a class="navbar-brand" href="/">  
 <img class="img-icon" src="{{ MEDIA\_URL }}image/icon.png">  
 </a>  
 </div>  
  
 <!-- Collect the nav links, forms, and other content for toggling -->  
 <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">  
 <ul class="nav navbar-nav navbar-right color-text">  
 <li><a class ="color-text" href="/create\_group/">Добавить кафедру</a></li>  
 {% if user.is\_authenticated %}  
 <li class="dropdown">  
 <a href="#" class="dropdown-toggle color-text color-nav" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{{ user.username }} <span class="caret"></span></a>  
 <ul class="dropdown-menu color-text color-nav">  
 <li><a href="/logout/">Выйти</a></li>  
 </ul>  
 </li>  
 {% else %}  
 <li class="dropdown">  
 <a href="#" class="dropdown-toggle color-text color-nav" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Вход или регистрация <span class="caret"></span></a>  
 <ul class="dropdown-menu color-text color-nav">  
 <li><a href="/login/"> Войти </a></li>  
 <li><a href="/register/">Регистрация</a></li>  
 </ul>  
 </li>  
  
 {% endif %}  
  
  
 </ul>  
 </div><!-- /.navbar-collapse -->  
 </div><!-- /.container-fluid -->  
 </nav>  
 <div class="size-osn">  
  
 {% for contact **in** contacts %}  
 {# Each "contact" is a Contact model object. #}  
 <div class="alert alert-success osn-color" role="alert">  
 <img class="img-size" src="{{ MEDIA\_URL }}{{ contact.picture }}">  
 <span class="color-ss"><a class="color-ss" href="{% url 'group\_url' contact.id %}">  
 {{ contact.name }}  
 </a></span>  
 </div>  
 {% empty %}  
 <li>пустой список</li>  
 {% endfor %}  
  
 <nav aria-label="...">  
 <ul class="pagination">  
 {% if contacts.has\_previous %}  
 <li class="active">  
 <span class="pag-color" >  
 <a class="pag-color" aria-hidden="true" href="?page={{ contacts.previous\_page\_number }}">&laquo;</a>  
 </span>  
 </li>  
 {% else %}  
 <li class="disabled">  
 <span class="pag-color" >  
 <span class="pag-color" aria-hidden="true">&laquo;</span>  
 </span>  
 </li>  
 {% endif %}  
 {% if contacts.paginator.num\_pages <= 4 %}  
 {% for j **in** range %}  
 <li class="active">  
 <span><a style="color:black **!important**;" aria-hidden="true" href="?page={{ j }}">{{ j }}</a><span class="sr-only">(current)</span></span>  
 </li>  
 {% endfor %}  
  
 {% else %}  
 <li class="active">  
 <span>1 <span class="sr-only">(current)</span></span>  
 </li>  
 <li class="active">  
 <span>...<span class="sr-only">(current)</span></span>  
 </li>  
 {% for j **in** range %}  
 <li class="active">  
 <span><a style="color:black **!important**;" aria-hidden="true" href="?page={{ j }}">{{ j }}</a><span class="sr-only">(current)</span></span>  
 </li>  
 {% endfor %}  
 <li class="active">  
 <span>...<span class="sr-only">(current)</span></span>  
 </li>  
 <li class="active">  
 <span>{{ contacts.paginator.num\_pages }} <span class="sr-only">(current)</span></span>  
 </li>  
 {% endif %}  
  
  
 {% if contacts.has\_next %}  
 <li class="active">  
 <span class="pag-color" >  
 <a class="pag-color" aria-hidden="true" href="?page={{ contacts.next\_page\_number }}">&raquo;</a>  
 </span>  
 </li>  
 {% else %}  
 <li class="disabled">  
 <span class="pag-color" >  
 <span aria-hidden="true">&raquo;</span>  
 </span>  
 </li>  
 {% endif %}  
  
 </ul>  
 </nav>  
 </div>  
{% endblock %}

Register.html

{% extends 'base.html' %}  
{% load static %}  
{% block title %}Регестрация{% endblock %}  
{% block body %}  
 <nav class="navbar navbar-default color-nav">  
 <div class="container-fluid">  
 <!-- Brand and toggle get grouped for better mobile display -->  
 <div class="navbar-header">  
 <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#bs-example-navbar-collapse-1" aria-expanded="false">  
 <span class="sr-only">Toggle navigation</span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 <span class="icon-bar"></span>  
 </button>  
 <a class="navbar-brand" href="/">  
 <img class="img-icon" src="{{ MEDIA\_URL }}/image/icon.png">  
 </a>  
 </div>  
  
 <!-- Collect the nav links, forms, and other content for toggling -->  
 <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">  
 <ul class="nav navbar-nav navbar-right color-text">  
<li><a class ="color-text" href="/create\_group/">Добавить кафедру</a></li>  
 {% if user.is\_authenticated %}  
 <li class="dropdown">  
 <a href="#" class="dropdown-toggle color-text color-nav" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">{{ user.username }} <span class="caret"></span></a>  
 <ul class="dropdown-menu color-text color-nav">  
 <li><a href="/logout/">Выйти</a></li>  
 </ul>  
 </li>  
 {% else %}  
 <li class="dropdown">  
 <a href="#" class="dropdown-toggle color-text color-nav" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">Вход или регистрация <span class="caret"></span></a>  
 <ul class="dropdown-menu color-text color-nav">  
 <li><a href="/login/"> Войти </a></li>  
 <li><a href="/register/">Регистрация</a></li>  
 </ul>  
 </li>  
  
 {% endif %}  
  
  
 </ul>  
 </div><!-- /.navbar-collapse -->  
 </div><!-- /.container-fluid -->  
 </nav>  
 <div>  
 <form class="form-signin" method="POST">  
 {% csrf\_token %}  
 {% if form.non\_field\_errors %}  
 {% for error **in** form.non\_field\_errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 <div class="form-group row">  
 <label for="example-text-input" class="col-xs-2 col-form-label">Логин</label>  
 <div class="col-xs-10">  
 {{ form.username }}  
 </div>  
 </div>  
 {% if form.username.errors %}  
 {% for error **in** form.username.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
  
 {% endfor %}  
 {% endif %}  
 <div class="form-group row">  
 <label for="example-password-input" class="col-xs-2 col-form-label">Пароль</label>  
 <div class="col-xs-10">  
 {{ form.password1 }}  
 </div>  
 </div>  
 {% if form.password1.errors %}  
 {% for error **in** form.password1.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 <div class="form-group row">  
 <label for="example-password-input" class="col-xs-2 col-form-label">Повторите пароль</label>  
 <div class="col-xs-10">  
 {{ form.password2 }}  
 </div>  
 </div>  
 {% if form.password2.errors %}  
 {% for error **in** form.password2.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
  
 <div class="form-group row">  
 <label for="example-email-input" class="col-xs-2 col-form-label">Email</label>  
 <div class="col-xs-10">  
 {{ form.email }}  
 </div>  
 </div>  
 {% if form.email.errors %}  
 {% for error **in** form.email.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 <div class="form-group row">  
 <label for="example-email-input" class="col-xs-2 col-form-label">Фамилия</label>  
 <div class="col-xs-10">  
 {{ form.last\_name }}  
 </div>  
 </div>  
 {% if form.last\_name.errors %}  
 {% for error **in** form.last\_name.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 <div class="form-group row">  
 <label for="example-email-input" class="col-xs-2 col-form-label">Имя</label>  
 <div class="col-xs-10">  
 {{ form.first\_name }}  
 </div>  
 </div>  
 {% if form.first\_name.errors %}  
 {% for error **in** form.first\_name.errors %}  
 <div class="error">  
 {{ error|escape }}  
 </div>  
 {% endfor %}  
 {% endif %}  
 <button class="btn btn-lg btn-primary btn-block" type="submit">Зарегистрироваться</button>  
 </form>  
 </div>  
{% endblock %}

Результаты









