ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл	

СОГЛАСОВАНО

Доцент департамента программной инженерии факультета компьютерных наук

_____ Х. М. Салех
«_____» _____ 2020 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия» профессор департамента программной инженерии, канд. техн. наук

_____ В. В. Шилов
«_____» _____ 2020 г.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СООБЩЕНИЙ О БЫТОВЫХ ПРОБЛЕМАХ СТУДЕНТОВ В ОБЩЕЖИТИИ

Сервер
Текст программы

Лист Утверждения

RU.17701729.02.07-01 12 01-1-ЛУ

Исполнитель: Студент группы БПИ-194
_____ В. А. Анненков
«_____» _____ 2020 г.

Москва 2020

Утверждён
RU.17701729.02.07-01 12 01-1-ЛУ

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ СООБЩЕНИЙ О БЫТОВЫХ ПРОБЛЕМАХ СТУДЕНТОВ В ОБЩЕЖИТИИ

Сервер
Текст программы

RU.17701729.02.07-01 12 01-1

Листов 27

Инв. № подл	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Москва 2020

Содержание

1. Текст программы

1.1. admin.py

```
from django.contrib import admin

from api import models

admin.site. site_title  = 'ВШЭ'
admin.site.site_header = 'Администрирование общежитий ВШЭ'


@admin.register(models.Problem)
class  ProblemAdmin(admin.ModelAdmin):
    list_display  = (
        ' title ',
        'author',
        'status',
        'created_at',
    )
    search_fields  = (
        ' title ',
        'description',
    )
    readonly_fields = (
        'created_at',
        'updated_at',
    )


@admin.register(models.Message)
class  MessageAdmin(admin.ModelAdmin):
    list_display  = (
        'text',
        'author',
        'problem',
        'dormitory',
        'is_read',
        'is_from_student',
        'created_at',
    )
    search_fields  = (
        'text',
        'author',
        'problem',
    )
     list_filter   = (
        'is_read',
```

```
        'is_from_student',
    )
    readonly_fields = (
        'created_at',
        'updated_at',
    )


@admin.register(models.Dormitory)
class DormitoryAdmin(admin.ModelAdmin):
    list_display = (
        'name',
        'address',
    )
    search_fields = (
        'name',
        'address',
    )


@admin.register(models.Notice)
class NoticeAdmin(admin.ModelAdmin):
    list_display = (
        'main_text',
        'is_important',
        'created_at',
    )
    search_fields = (
        'main_text',
        'text',
    )
     list_filter = (
        'is_important',
    )
    readonly_fields = (
        'created_at',
        'updated_at',
    )


@admin.register(models.Event)
class EventAdmin(admin.ModelAdmin):
    list_display = (
        ' title ',
        'target_date',
        'created_at',
    )
    search_fields = (
```

```
        'title',
        'description',
    )
    readonly_fields = (
        'created_at',
        'updated_at',
    )


@admin.register(models.Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = (
        'user',
        'dormitory',
        'is_login',
        'is_accept',
    )
    search_fields = (
        'user',
        'dormitory',
    )
     list_filter = (
        'dormitory',
        'is_login',
        'is_accept',
    )


@admin.register(models.Confirmation)
class ConfirmationAdmin(admin.ModelAdmin):
    list_display = (
        'email',
        'code',
    )
    search_fields = (
        'email',
        'code',
    )
```

1.2. apps.py

```
from django.apps import AppConfig


class ApiConfig(AppConfig):
    name = 'api'
```

1.3. exceptions.py

```
from rest_framework import status
from rest_framework.exceptions import APIException


class Unauthorized(APIException):
    status_code = status.HTTP_401_UNAUTHORIZED
    default_detail = 'Для просмотра этой страницы требуется авторизация.'
    default_code = 'unauthorized'


class WrongEmail(Exception):
    status_code = status.HTTP_401_UNAUTHORIZED
    default_detail = 'Этот Email не принадлежит ни одному студентуработнику/.'


class CodeConfirmationException(Exception):
    status_code = status.HTTP_401_UNAUTHORIZED
    default_detail = 'Неверный код подтверждения.'
```

1.4. models.py

```
from django.contrib.auth.models import User
from django.db import models
from django.db.models import signals
from django.dispatch import receiver


class Dormitory(models.Model):
    name = models.CharField(verbose_name='имя общежития',
                            max_length=200)
    address = models.CharField(verbose_name='адрес',
                               max_length=500)

    class Meta:
        verbose_name = 'общежитие'
        verbose_name_plural = 'общежития'

    def __str__(self):
        return self.name


class Problem(models.Model):
    class Status(models.TextChoices):
        OPEN = 'O', 'Открыта'
        AGENT_REPLY_PROCESS = 'ARP', 'Ожидание ответа'
        RESOLVING = 'R', 'Проблема решается'
        CLOSED = 'C', 'Закрыта'

    author = models.ForeignKey(verbose_name='автор',
```

```
                                    to=User,
                                    on_delete=models.CASCADE)
    title  = models.CharField(verbose_name='заголовок',
                                  max_length=100)
    description  = models.TextField(verbose_name='описание проблемы',
                                      max_length=10000)
    status  = models.CharField(verbose_name='статус',
                                  max_length=5,
                                  choices=Status.choices,
                                  default=Status.OPEN)

    created_at = models.DateTimeField(verbose_name='дата создания',
                                      auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name='дата обновление',
                                      auto_now=True)

    class Meta:
        verbose_name = 'обращение'
        verbose_name_plural = 'обращения'

    def __str__(self):
        return '{}: {}'.format(self.author, self.title[:17])


class Message(models.Model):
    author = models.ForeignKey(verbose_name='автор',
                                  to=User,
                                  on_delete=models.CASCADE)
    text  = models.TextField(verbose_name='описание проблемы',
                                max_length=10000)
    is_read = models.BooleanField(verbose_name='прочитано',
                                    default=False)
    is_from_student = models.BooleanField(verbose_name='от студента ли',
                                          default=True)
    problem = models.ForeignKey(verbose_name='обращение',
                                   to=Problem,
                                   null=True,
                                   related_name='messages',
                                   blank=True,
                                   on_delete=models.CASCADE)
    dormitory = models.ForeignKey(verbose_name='общежитие',
                                     to=Dormitory,
                                     null=True,
                                     related_name='messages',
                                     blank=True,
                                     on_delete=models.CASCADE)

    created_at = models.DateTimeField(verbose_name='дата создания',
```

```
                                        auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name='дата обновление',
                                        auto_now=True)

    class Meta:
        verbose_name = 'сообщение'
        verbose_name_plural = 'сообщения'

    def __str__(self):
        return '{}: {}'.format(self.author, self.text[:17])


class Notice(models.Model):
    main_text = models.CharField(verbose_name='главный текст',
                                 max_length=200)
    text = models.TextField(verbose_name='текст',
                            max_length=10000)
    is_important = models.BooleanField(verbose_name='важное ли',
                                       default=False)

    created_at = models.DateTimeField(verbose_name='дата создания',
                                      auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name='дата обновление',
                                      auto_now=True)

    class Meta:
        verbose_name = 'объявление'
        verbose_name_plural = 'объявления'

    def __str__(self):
        return '{}'.format(self.main_text[:25])


class Event(models.Model):
    title = models.CharField(verbose_name='заголовок',
                             max_length=100)
    description = models.TextField(verbose_name='описание',
                                   max_length=10000)

    target_date = models.DateTimeField(verbose_name='дата проведения')

    created_at = models.DateTimeField(verbose_name='дата создания',
                                      auto_now_add=True)
    updated_at = models.DateTimeField(verbose_name='дата обновление',
                                      auto_now=True)

    class Meta:
        verbose_name = 'событие'
```

```
        verbose_name_plural = 'события'

    def __str__(self):
        return self.title


class Profile(models.Model):
    class Role(models.TextChoices):
        STUDENT = 'student', 'Студент'
        AGENT = 'agent', 'Агент поддержки'

    user = models.OneToOneField(verbose_name='пользователь',
                                to=User,
                                on_delete=models.CASCADE)
    dormitory = models.ForeignKey(verbose_name='общежитие',
                                  to=Dormitory,
                                  null=True,
                                  blank=True,
                                  on_delete=models.CASCADE)
    room = models.CharField(verbose_name='комната',
                            null=True,
                            blank=True,
                            max_length=20)

    role = models.CharField(verbose_name='роль',
                            max_length=20,
                            choices=Role.choices,
                            default=Role.STUDENT)

    is_login = models.BooleanField(verbose_name='был ли первый вход',
                                   default=False)
    is_accept = models.BooleanField(verbose_name='подтверждён ли администратором',
                                    default=False)

    class Meta:
        verbose_name = 'профиль'
        verbose_name_plural = 'профили'

    def __str__(self):
        return self.user.email


class Confirmation(models.Model):
    email = models.CharField(verbose_name='Email',
                             max_length=40)
    code = models.CharField(verbose_name='код подтверждения',
                            max_length=10)
```

```
    class Meta:
        verbose_name = 'подтверждение'
        verbose_name_plural = 'подтверждения'

    def __str__(self):
        return '{} — {}'.format(self.email, self.code)


@receiver(signals.post_save, sender=User)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)


@receiver(signals.post_save, sender=User)
def save_user_profile(sender, instance, **kwargs):
    instance.profile.save()
```

1.5. serializers.py

```
from django.contrib.auth.backends import UserModel
from django.contrib.auth.models import User
from rest_framework import serializers

from api import models


class UserSerializer(serializers.HyperlinkedModelSerializer):
    password = serializers.CharField(write_only=True)

    def create(self, validated_data):
        user = UserModel.objects.create(
            username=validated_data['username']
        )
        user.set_password(validated_data['password'])
        user.save()

        return user

    class Meta:
        model = User
        fields = [
            'id',
            'url',
            'username',
            'password',
            'email',
            'groups'
        ]
```

```
class DormitorySerializer( serializers .HyperlinkedModelSerializer):
    messages = serializers .SerializerMethodField()

    class Meta:
        model = models.Dormitory
        fields = [
            'id',
            'name',
            'address',
            'messages'
        ]

    def get_messages(self, obj):
        ordered_queryset = reversed(models.Message.objects.filter(dormitory_id=obj.id).
            order_by('-id')[:30])
        return MessageSerializer(ordered_queryset, many=True, context=self.context).data


class ProblemSerializer( serializers .ModelSerializer):
    messages = serializers .SerializerMethodField()
    author_first_name = serializers.CharField(source='author.first_name')
    author_last_name = serializers.CharField(source='author.last_name')

    class Meta:
        model = models.Problem
        fields = [
            'id',
            'author',
            'author_first_name',
            'author_last_name',
            'title',
            'description',
            'status',
            'created_at',
            'updated_at',
            'messages',
        ]

    def get_messages(self, obj):
        ordered_queryset = models.Message.objects.filter(problem_id=obj.id).order_by('id')
        return MessageSerializer(ordered_queryset, many=True, context=self.context).data


class MessageSerializer( serializers .ModelSerializer):
    author_first_name = serializers.CharField(source='author.first_name')
    author_last_name = serializers.CharField(source='author.last_name')
```

```
    class Meta:
        model = models.Message
        fields = [
            'id',
            'author',
            'author_first_name',
            'author_last_name',
            'text',
            'is_read',
            'is_from_student',
            'created_at',
            'updated_at'
        ]


class NoticeSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = models.Notice
        fields = [
            'id',
            'main_text',
            'text',
            'is_important',
            'created_at',
            'updated_at'
        ]


class EventSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = models.Event
        fields = [
            'id',
            'title',
            'description',
            'target_date',
            'created_at',
            'updated_at'
        ]
```

1.6. views.py

```
import asyncio
import random
import string

import aioruz
from django.contrib.auth.models import User
```

```
from django.core.mail import send_mail
from rest_framework import viewsets, status, views
from rest_framework.authtoken.models import Token
from rest_framework.response import Response

from api import models, serializers , exceptions
from api.exceptions import WrongEmail, CodeConfirmationException, Unauthorized


def get_rnd(length=10):
    return ''.join(random.SystemRandom().choices(string.ascii_letters + string.digits, k=length)
        )


async def get_student_by_email_async(email):
    try:
        student = await aioruz.student_info(email)
        return student
    except LookupError:
        return None


def get_student_by_email(email):
    loop = asyncio.new_event_loop()
    result = loop.run_until_complete(get_student_by_email_async(email))
    return result


class AuthView(views.APIView):
    def post(self, request):
        try:
            email = request.data['email']
            if email == 'zzoorm@gmail.com':
                return Response(status=status.HTTP_200_OK)

            student = get_student_by_email(email)
            if student is None:
                raise WrongEmail

            first_name = student['fio'].split('␣')[1]
            last_name = student['fio'].split('␣')[0]

            code = get_rnd(6)
            models.Confirmation.objects.create(
                email=email,
                code=code,
            )
```

```
            if len(models.User.objects.filter(email=email)) == 0:
                User.objects.create_user(
                    email=email,
                    username=email,
                    first_name=first_name,
                    last_name=last_name,
                    password='thisisejkflkwefmlkwlkefklkjwenf21829834hadrkey'
                )

            send_mail('Код подтверждения',
                      'Ваш код подтверждения: {}'.format(code),
                      'confrim@hsesupporter.ru',
                      [email],
                       fail_silently=False)

            return Response(
                {
                    'message': 'Код подтверждения отправлен на почту.',
                    'profile': {
                        'first_name': first_name,
                        'last_name': last_name,
                        'info': student['info'],
                        'role': 'student',
                    },
                },
                status=status.HTTP_200_OK)

        except WrongEmail as e:
            return Response({'message': e.default_detail}, status=e.status_code)

        except KeyError as e:
            return Response({'message': str(e)}, status=status.HTTP_400_BAD_REQUEST)

        except Exception as e:
            return Response({'message': str(e)}, status=status.HTTP_400_BAD_REQUEST)


class AuthConfirmView(views.APIView):
    def post(self, request):
        try:
            email = request.data['email']
            code = request.data['code']

            if email == 'zzoorm@gmail.com':
                user = models.User.objects.get(email=email)
                token = list(Token.objects.filter(user=user))[0].key
                return Response(
                    {
```

```
                'message': 'Успешная авторизация.',
                'is_accept': user.profile.is_accept,
                'token': token,
                'profile': {
                    'id': user.id,
                    'fio': 'Евгеньев Евгений Евгеньевич',
                    'info': 'Босс',
                    'role': 'agent',
                },
            }, status=status.HTTP_200_OK)

    student = get_student_by_email(email)
    if student is None:
        raise WrongEmail

    if len(models.Confirmation.objects.filter(email=email, code=code)) == 0:
        raise CodeConfirmationException
    confirmation = list(models.Confirmation.objects.filter(email=email, code=code))[0]
    # confirmation.delete()

    user = models.User.objects.get(email=email)
    if user is None:
        raise CodeConfirmationException
    user.profile.is_login = True
    user.save()

    if len(Token.objects.filter(user=user)) == 0:
        token = get_rnd(40)
        Token.objects.create(
            user=user,
            key=token,
        )
    else:
        token = list(Token.objects.filter(user=user))[0].key

    return Response(
        {
            'message': 'Успешная авторизация.',
            'is_accept': user.profile.is_accept,
            'token': token,
            'profile': {
                'id': user.id,
                'first_name': user.first_name,
                'last_name': user.last_name,
                'info': student['info'],
                'role': 'student',
            },
        }, status=status.HTTP_200_OK)
```

```
        except CodeConfirmationException as e:
            return Response({'message': e.default_detail}, status=e.status_code)

        except WrongEmail as e:
            return Response({'message': e.default_detail}, status=e.status_code)

        except KeyError as e:
            return Response({'message': str(e)}, status=status.HTTP_400_BAD_REQUEST)

        except Exception as e:
            return Response({'message': str(e)}, status=status.HTTP_400_BAD_REQUEST)


class ProfileView(views.APIView):
    def get(self, request):
        try:
            include_additional_info = False
            if 'include_additional_info' in request.query_params:
                include_additional_info = bool(request.query_params['include_additional_info'
                    ])

            user = request.user
            if user.id is None:
                raise Unauthorized

            dormitory_name = None
            if user.profile.dormitory is not None:
                dormitory_name = str(user.profile.dormitory.name)

            dormitory_address = None
            if user.profile.dormitory is not None:
                dormitory_address = str(user.profile.dormitory.address)

            additional_info = None
            if True:
                additional_info = {
                    'dormitory_users_count': len(models.Profile.objects.filter(dormitory_id=
                        user.profile.dormitory_id)),
                    'agents_online_count': 1,
                }

            return Response(
                {
                    'id': user.id,
                    'first_name': user.first_name,
                    'last_name': user.last_name,
                    'role': user.profile.role,
```

```
                'room': user.profile.room,
                'dormitory': {
                    'id': user.profile.dormitory_id,
                    'name': dormitory_name,
                    'address': dormitory_address,
                },
                'additional_info': additional_info,
            }, status=status.HTTP_200_OK)

        except Unauthorized as e:
            return Response({
                'message': e.default_detail
            }, status=e.status_code)

    def post(self, request):
        try:
            dormitory_name = request.data['dormitory']
            room = request.data['room']

            user = request.user
            if user.id is None:
                raise Unauthorized

            dormitory = models.Dormitory.objects.get(name=dormitory_name)

            if user.profile.dormitory.name != dormitory.name:
                user.profile.dormitory = dormitory
            if user.profile.room != room:
                user.profile.room = room

            user.save()

            return Response({
                'id': user.id,
                'first_name': user.first_name,
                'last_name': user.last_name,
                'role': user.profile.role,
                'room': user.profile.room,
                'dormitory': {
                    'id': user.profile.dormitory_id,
                    'name': dormitory.name,
                    'address': dormitory.address,
                },
            }, status=status.HTTP_200_OK)

        except KeyError as e:
            return Response({
                'message': str(e)
```

```
            }, status=status.HTTP_400_BAD_REQUEST)

        except Unauthorized as e:
            return Response({
                'message': e.default_detail
            }, status=e.status_code)

        except Exception as e:
            return Response({
                'message': str(e)
            }, status=status.HTTP_400_BAD_REQUEST)


class MainPageView(views.APIView):
    def get(self, request):
        try:
            user = request.user
            if user.id is None:
                raise Unauthorized

            dormitory_name = None
            if user.profile.dormitory is not None:
                dormitory_name = str(user.profile.dormitory.name)

            dormitory_address = None
            if user.profile.dormitory is not None:
                dormitory_address = str(user.profile.dormitory.address)

            profile = {
                'id': user.id,
                'first_name': user.first_name,
                'last_name': user.last_name,
                'role': user.profile.role,
                'room': user.profile.room,
                'dormitory': {
                    'id': user.profile.dormitory_id,
                    'name': dormitory_name,
                    'address': dormitory_address,
                },
            }

            notice_list = models.Notice.objects.all().order_by('-created_at')[:10]
            notices = [{
                'id': notice.id,
                'main_text': notice.main_text,
                'text': notice.text,
                'is_important': notice.is_important,
                'created_at': notice.created_at,
```

```
            'updated_at': notice.updated_at
        } for notice in notice_list]

        event_list = models.Event.objects.all().order_by('target_date')[:3]
        events = [{
            'id': event.id,
            'title': event.title,
            'description': event.description,
            'target_date': event.target_date,
            'created_at': event.created_at,
            'updated_at': event.updated_at
        } for event in event_list]

        main_questions = [
            {
                'question': 'Как пообедать?',
                'answer': 'Для этого нужно получить талон на еду.'
            },
            {
                'question': 'Как поспать?',
                'answer': 'Для этого нужно лечь в кровать и уснуть.'
            },
            {
                'question': 'Как работать в карантин?',
                'answer': 'Для этого нужно взять себя и свой ноутбук в руки.'
            }
        ]

        additional_info = {}

        return Response(
            {
                'profile': profile,
                'notices': notices,
                'events': events,
                'main_questions': main_questions,
                'additional_info': additional_info,
            }, status=status.HTTP_200_OK)

    except Unauthorized as e:
        return Response({
            'message': e.default_detail
        }, status=e.status_code)


class AcceptStatusView(views.APIView):
    def get(self, request):
        try:
```

```
            user = request.user

            if user.id is None:
                raise Unauthorized

            return Response({
                'is_accept': user.profile.is_accept
            }, status=status.HTTP_200_OK)

        except Unauthorized as e:
            return Response({
                'message': e.default_detail
            }, status=e.status_code)


class DormitoriesViewSet(viewsets.ModelViewSet):
    serializer_class = serializers.DormitorySerializer
    http_method_names = ['get', 'head']

    def get_queryset(self):
        try:
            return models.Dormitory.objects.all()
        except TypeError:
            raise exceptions.Unauthorized()

    def get(self, request):
        if request.user is not None:
            return Response(self.serializer_class.data)
        else:
            return Response(status=status.HTTP_401_UNAUTHORIZED)


class ProblemViewSet(viewsets.ModelViewSet):
    serializer_class = serializers.ProblemSerializer

    def get_queryset(self):
        try:
            if self.request.user.groups.filter(name='Агент_поддержки').exists():
                return models.Problem.objects.all().order_by('-id')
            return models.Problem.objects.all().order_by('-id').filter(author=self.request.user.
                id)
        except TypeError:
            raise exceptions.Unauthorized()

    def get(self, request):
        return Response(self.serializer_class.data)

    def create(self, request, *args, **kwargs):
```

```
        try:
            if request.data['status'] is None:
                request.data['status'] = models.Problem.Status.OPEN
            if request.data['title'] is None or request.data['description'] is None:
                raise exceptions.APIException

            models.Problem.objects.create(
                author=request.user,
                title=request.data['title'],
                description=request.data['description'],
                status=request.data['status']
            )

            return Response(status=status.HTTP_201_CREATED)

        except Exception:
            return Response(status=status.HTTP_400_BAD_REQUEST)

    def destroy(self, request, *args, **kwargs):
        try:
            pk = int(kwargs['pk'])
            author = request.user

            models.Problem.objects.get(id=pk).delete()
            return Response(status=status.HTTP_200_OK)

        except Exception:
            return Response(status=status.HTTP_400_BAD_REQUEST)


class MessagesViewSet(viewsets.ModelViewSet):
    serializer_class = serializers.MessageSerializer

    def get_queryset(self):
        try:
            return models.Message.objects.all().order_by('id')
        except TypeError:
            raise exceptions.Unauthorized()

    def get(self, request):
        return Response(self.serializer_class.data)

    def create(self, request, *args, **kwargs):
        try:
            problem = None
            if 'problem' in request.data and request.data['problem'] != 0:
                problem = request.data['problem']
```

```
            dormitory = None
            if 'dormitory' in request.data and request.data['dormitory'] != 0:
                dormitory = request.data['dormitory']

            models.Message.objects.create(
                author=request.user,
                text=request.data['text'],
                problem_id=problem,
                dormitory_id=dormitory
            )

            return Response(status=status.HTTP_201_CREATED)

        except Exception as ex:
            return Response({"Fail": ex}, status=status.HTTP_400_BAD_REQUEST)


class NoticesViewSet(viewsets.ModelViewSet):
    serializer_class = serializers.NoticeSerializer

    def get_queryset(self):
        try:
            return models.Notice.objects.all().order_by('-id')
        except TypeError:
            raise exceptions.Unauthorized()

    def get(self, request):
        if request.user is not None:
            return Response(self.serializer_class.data)
        else:
            return Response(status=status.HTTP_401_UNAUTHORIZED)
```

1.7. settings.py

```
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__)))

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.0/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = '33%nnlgy0ma%v=4kh&iv=1(g-k&!*g2%p5g5v7$ukop-*87y_t'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = [
```

```
    'hse-app.herokuapp.com',
    'hse-supporter.herokuapp.com',
    '127.0.0.1',
]

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'api.apps.ApiConfig',
    'rest_framework.authtoken',
    'rest_framework',
    'djoser',
]

REST_FRAMEWORK = {
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.AllowAny',
    ],
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
}

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',
    'whitenoise.middleware.WhiteNoiseMiddleware',
]

ROOT_URLCONF = 'HSESupporterBackend.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',
            ],
        },
    },
]

WSGI_APPLICATION = 'HSESupporterBackend.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'd9c8qb2n2vtifk',
        'USER': 'rextircvkkxqdf',
        'PASSWORD': os.environ.get('DATABASE_PASSWORD', ''),
        'HOST': 'ec2-54-247-125-38.eu-west-1.compute.amazonaws.com',
        'PORT': '5432',
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.0/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/3.0/topics/i18n/
```

```
LANGUAGE_CODE = 'ru'

TIME_ZONE = 'Europe/Moscow'

USE_I18N = True

USE_L10N = True

USE_TZ = True

EMAIL_USE_TLS = True
EMAIL_HOST = 'smtp.gmail.com'
EMAIL_PORT = 587
EMAIL_HOST_USER = os.environ.get('EMAIL_LOGIN', '')
EMAIL_HOST_PASSWORD = os.environ.get('EMAIL_PASSWORD', '')
DEFAULT_FROM_EMAIL = 'confrim@hse.ru'
DEFAULT_TO_EMAIL = 'vaannenkov@edu.hse.rur'

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC_URL = '/static/'
BASE_DIR = os.path.dirname(os.path.dirname(__file__))
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
STATICFILES_DIRS = (
    os.path.join(BASE_DIR, 'static'),
)
TEMPLATE_DIRS = (
    os.path.join(BASE_DIR, 'templates'),
)

STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'
```

1.8. urls.py

```
from django.contrib import admin
from django.urls import path, include
from rest_framework import routers

from api import views

router = routers.DefaultRouter()

router.register(r'notices', views.NoticesViewSet, basename='Notices')
router.register(r'problems', views.ProblemViewSet, basename='Problems')
router.register(r'messages', views.MessagesViewSet, basename='Messages')
router.register(r'dormitories', views.DormitoriesViewSet, basename='Dormitories')
```

```
urlpatterns = [
    path('', include(router.urls)),
    path('admin/', admin.site.urls),

    path('auth/register/', views.AuthView.as_view()),
    path('auth/register/confirm-email/', views.AuthConfirmView.as_view()),
    path('auth/accept-status/', views.AcceptStatusView.as_view()),

    path('main_page/', views.MainPageView.as_view()),
    path('profile/', views.ProfileView.as_view()),

    path('auth/', include('djoser.urls.authtoken')),
]
```

Лист регистрации изменений

Изм.	Номера листов				Всего листов в документе	№ документа	Входящий № сопровод. докум. и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					