МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»

(БГТУ им. В.Г. Шухова)

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине «Базы данных»

тема «Разработка сайта городской администрации»

(наименование работы)

Автор работы: _____ Воскобойников И. С. ВТ-32

(подпись) (ФИО, группа)

Руководитель проекта: _____ Панченко М. В.

(подпись) (ФИО)

Оценка ___

Белгород 2020

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

Местная администрация (исполнительно-распорядительный орган муниципального образования) наделяется уставом муниципального образования полномочиями по решению вопросов местного значения и полномочиями для осуществления отдельных государственных полномочий, переданных органам местного самоуправления федеральными законами и законами субъектов Российской Федерации.

Местной администрацией руководит глава местной администрации на принципах единоначалия.

Главой местной администрации является глава муниципального образования либо лицо, назначаемое на должность главы местной администрации по контракту, заключаемому по результатам конкурса на замещение указанной должности на срок полномочий, определяемый уставом муниципального образования.

Местная администрация разрабатывает проекты бюджета, смет внебюджетных фондов, планов и программ социально-экономического развития города и представляет их на утверждение представительному органу.

Кроме того, администрация:

- исполняет бюджет и представляет на утверждение представительному органу отчет о его исполнении;

- обеспечивает функционирование муниципального жилищно-коммунального хозяйства и транспорта, муниципальных учреждений, здравоохранения, образования, культуры, других муниципальных учреждений;

- распоряжается и управляет муниципальной и иной переданной в управление города собственностью в порядке, установленном представительному органу;

- ведет территориальный кадастр, техническую инвентаризацию движимого и недвижимого имущества;

- разрабатывает предложения о создании, реорганизации и ликвидации муниципальных предприятий и учреждений;

- утверждает уставы муниципальных предприятий, организаций, учреждений;

- осуществляет исполнение государственных полномочий, переданных органам местного самоуправления города;

- выполняет в случаях и порядке, установленных представительном органом, функции заказчика по муниципальным контрактам на выполнение подрядных работ (оказание услуг, поставку товаров) для нужд муниципального образования города;

- в случаях, установленных представительным органом, готовит и представляет на его рассмотрение проекты правовых актов;

- создает в установленном порядке рабочие группы и коллегии, консультативные общественные и экспертные советы, привлекает на договорной основе научные организации, ученых и специалистов к решению проблем по вопросам местного значения;

- взаимодействует с органами государственной власти, местного самоуправления, предприятиями, учреждениями, организациями, а также с должностными лицами и гражданами по предметам своего ведения;

- участвует в разработке проектов соглашений, договоров муниципального образования города с другими муниципальными образованиями, органами исполнительной власти по вопросам местного значения и др.

2. ОСНОВНАЯ ЧАСТЬ

2.1 ПОСТАНОВКА ЗАДАЧИ

Разработать сайт городской администрации. Предусмотреть наличие страниц «Новости», «Анонс мероприятий», «Сотрудники», «Обращения граждан».

Страница «Новости» должна содержать список новостей. Вся информация должна хранится в спроектированной базе данных и динамически подгружаться на сайт. Пользователь должен иметь возможность выбирать новость для просмотра детальной информации.

Страница «Анонс мероприятий» должна содержать список мероприятий. Вся информация должна хранится в спроектированной базе данных и динамически подгружаться на сайт. Пользователь должен иметь возможность выбирать мероприятие для просмотра детальной информации.

Страница «Сотрудники» должна содержать список сотрудников. Вся информация должна хранится в спроектированной базе данных и динамически подгружаться на сайт. Пользователь должен иметь возможность выбирать сотрудника для просмотра детальной информации.

Страница «Обращения граждан» должна содержать форму для обратной связи. Пользователь должен иметь возможность отравлять свои обращения.

При этом пользователь также должен иметь возможность регистрации и авторизации на сайте.

Предусмотреть разграничение прав пользователей сайта (от двух групп пользователей и более).

Предусмотреть возможность бэкапа базы данных на яндекс диск.
Предусмотреть возможность выгрузки базы данных в форматах json и xml.
Автоматизировать процесс выгрузки копии базы данных ежедневно на Яндекс.Диск.

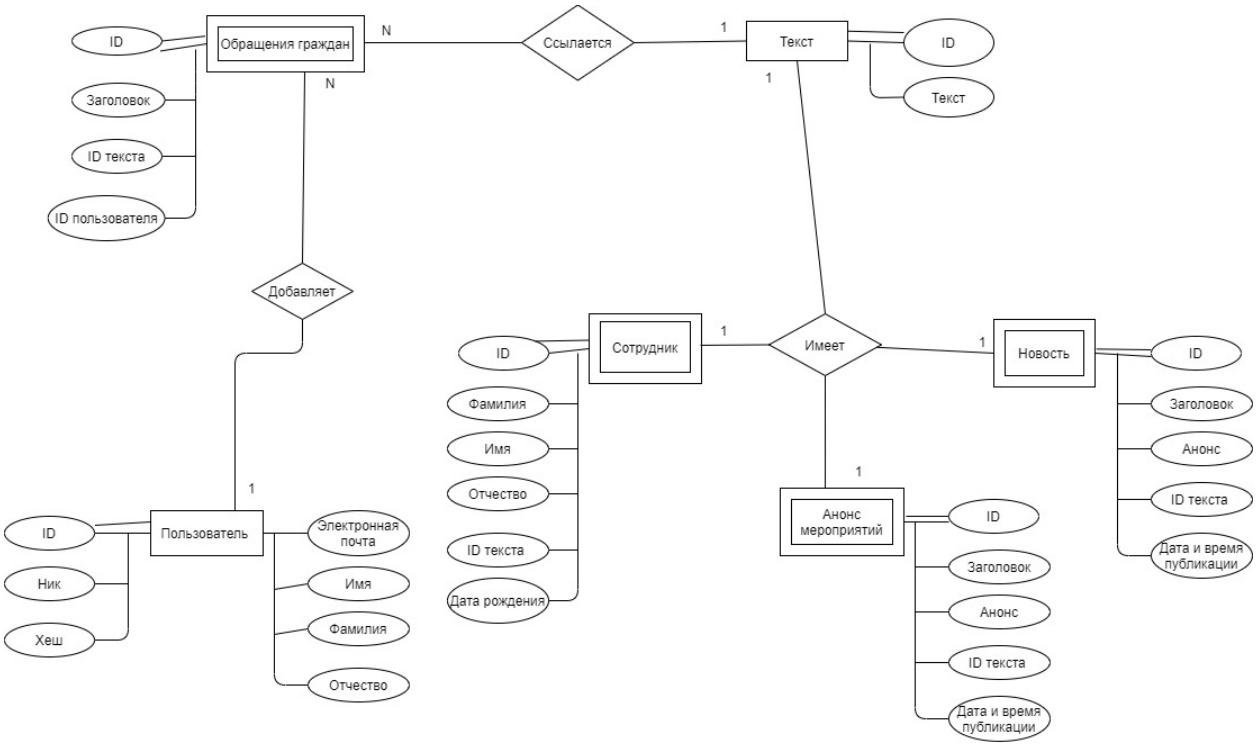
2.2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

2.2.1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

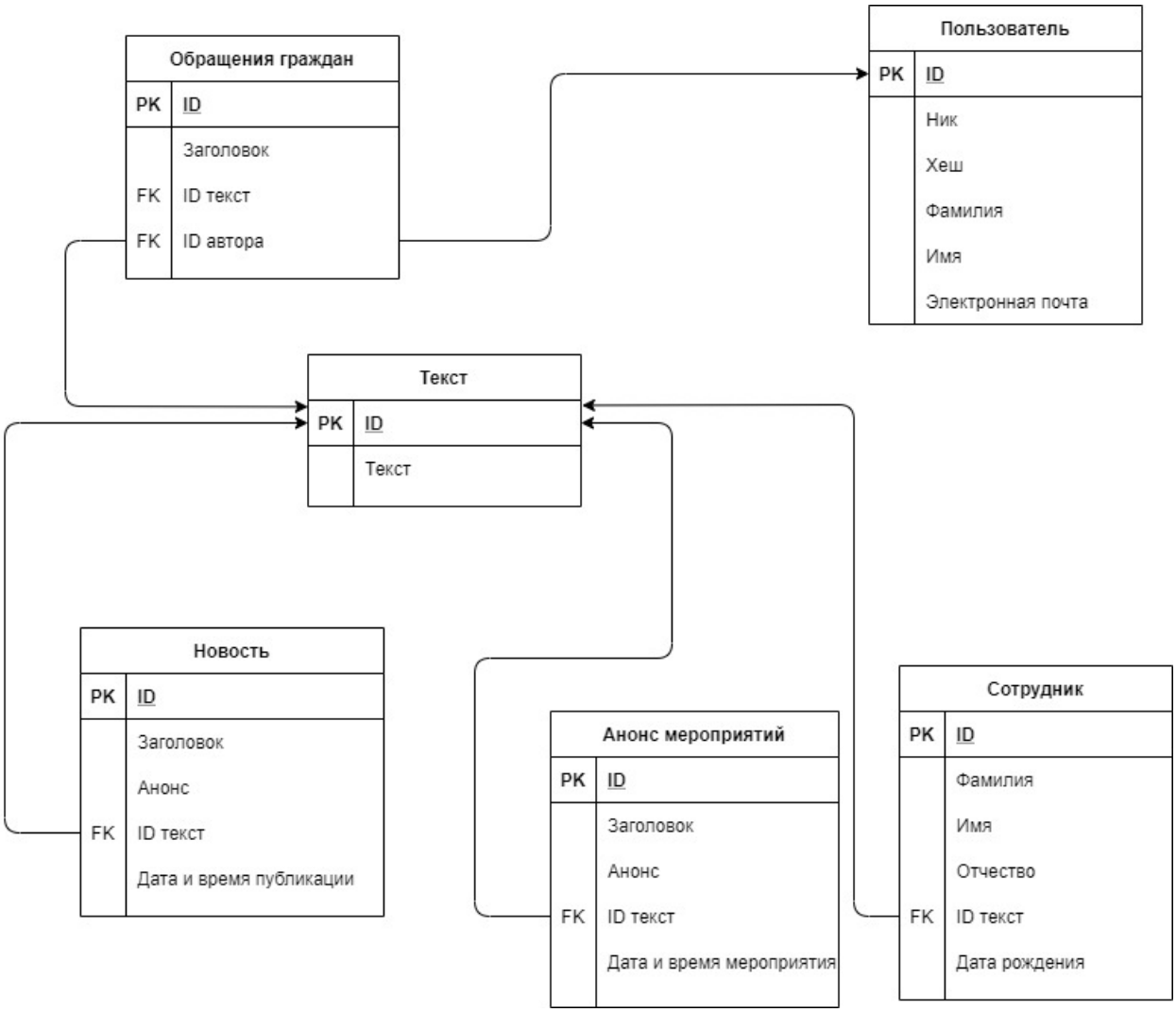
В процессе анализа предметной области были выделены следующие сущности:

1) НОВОСТЬ – информация о событии. Новость имеет следующие атрибуты: ID, заголовок, анонс, описание и дату и время публикации.
2) ТЕКСТ – подробная информация о новости, мероприятии или работнике. Описание имеет следующие атрибуты: ID, текст.
3) АНОНС МЕРОПРИЯТИЯ – информация о проводимых мероприятиях. Анонс мероприятия имеет следующие атрибуты: ID, заголовок, анонс, описание и дату и время проведения.
4) АКТЕР – человек, снимавшийся в фильмах киностудии. Актер имеет следующие атрибуты: ID, имя, фамилию, отчество, портрет, описание и дату рождения
5) СОТРУДНИК – человек, который работает в администрации. Сотрудник имеет следующие атрибуты: ID, имя, фамилию, отчество, описание и дату рождения.
6) ПОЛЬЗОВАТЕЛЬ – лицо, зарегистрированное на сайте. Пользователь имеет следующие атрибуты: ID, ник, пароль, адрес электронной почты, имя и фамилию

2.2.2 ДИАГРАММА «СУЩНОСТЬ - СВЯЗЬ»



2.2.3 СХЕМА СТРУКТУРЫ БАЗЫ ДАННЫХ



2.2.4 ОПИСАНИЕ АТРИБУТОВ ОТНОШЕНИЙ

News

1) ID: целочисленный, PK
2) title: строковый
3) anons: строковый
4) text: целочисленный, FK на Text
5) date: дата

Workers

1) ID: целочисленный, PK
2) surname: строковый
3) name: строковый
4) middlename: строковый
5) text: целочисленный, FK на Text
6) date: дата

Events

1) ID: целочисленный, PK
2) title: строковый
3) anons: строковый
4) text: целочисленный, FK на Text
5) date: дата

Text

1) ID: целочисленный, PK
2) text: строковый

Refences

1) ID: целочисленный, PK
2) title: строковый
3) text: целочисленный, FK на Text
4) user: целочисленный, FK на User

User

1) ID: целочисленный, PK
2) username: строковый
3) password: строковый
4) email: строковый, необязательный
5) first_name: строковый, необязательный
6) last_name: строковый, необязательный

2.2.5. НОРМАЛИЗАЦИЯ БАЗЫ ДАННЫХ

БД соответствует первой нормальной форме, так как все входящие в отношение атрибуты являются атомарными (неделимыми).

БД соответствует второй нормальной форме, так как находится в первой нормальной форме и каждый неключевой атрибут отношения функционально зависит от первичного ключа.

БД соответствует третьей нормальной форме, так как находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Описание функциональных зависимостей

News

1) ID -> title
2) ID -> anons
3) ID -> text
4) ID -> date

Workers

1) ID -> surname
2) ID -> name
3) ID -> middlename
4) ID -> text
5) ID -> date

Events

1) ID -> title
2) ID -> anons
3) ID -> text
4) ID -> date

Text

1) ID -> text

Refences

1) ID -> title

2) ID -> text
3) ID -> user

User

1) ID -> username
2) ID -> password
3) ID -> email
4) ID -> first_name
5) ID -> last_name

2.2.6 ОПИСАНИЕ ПРОЦЕССА СОЗДАНИЯ БАЗЫ ДАННЫХ

При создании базы данных были использованы возможности фреймворка Django – классы-модели. Модели определяют структуру хранимых данных, включая типы полей и, возможно, их максимальный размер, значения по умолчанию, параметры списка выбора, текст справки для документации, текст меток для форм и т. д. Определение модели не зависит от основной базы данных. После того, как была выбрана конкретная база данных, с ней не нужно работать напрямую - достаточно просто написать свою структуру модели и код, а Django организует работу, связанную с базой данных.

Для создания базы данных с помощью фреймворка Django сначала выберем БД, с которой будем работать. В проекте будем использовать базу данных PostgreSQL. Для этого в файле settings.py проекта пропишем следующий код:

settings.py

```
DATABASES = {
    'default': {
    'ENGINE': 'django.db.backends.postgresql_psycopg2',
    'NAME': 'site',
    'USER': 'postgres',
    'PASSWORD': '123456789',
    'HOST': 'localhost',
    'PORT': '5432'
    }
}
```

Теперь можно приступать непосредственно к созданию моделей Django. Для этого в файле нашего приложения models.py необходимо создать класс-наследник от класса Models, хранящегося в библиотеке фреймворка по пути django.db. Рассмотрим процесс создания моделей на примере создания модели «Новости» News.

models.py

```
class News(models.Model):
    title = models.CharField('Название', max_length=50)
    anons = models.CharField('Анонс', max_length=250)
    text = models.ForeignKey(
        Text, related_name='news', on_delete=models.CASCADE, null=False,
verbose_name='Текст'
    )
    date = models.DateTimeField('Дата публикации')

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = 'Новость'
        verbose_name_plural = 'Новости'
        ordering = ['-date']
```

Название класса становится названием таблицы в базе данных. При определении класса News наследника Models в нашей базе данных будет создана таблица news Ее атрибутами станут поля класса News, они будут иметь такие же типы, которые присваиваются полям. Например, если полю title присваивается класс CharField, то атрибут с идентичным названием таблицы actor будет иметь тип varchar. Параметр max_length в данном случае накладывает ограничение на максимальную длину строки. В SQL это эквивалентно типу varchar(max_length).

Для реализации связи между таблицами «один-к-одному» и «один-ко-многим» используется класс ForeignKey. Параметрами его конструктора являются класс модели, на которую ссылается создаваемая модель, и правило обеспечение ссылочной целостности. Например, модель News ссылается на модель Text и при удалении экземпляра «описания», на который ссылается «новость», будет также удален и его экземпляр, потому что параметр on_delete имеет значение CASCADE.

Вложенный класс Meta мы используем для настройки отображения информации о создаваемой таблице. Например, verbose_name = ‘Новость’ означает, что название таблицы при работе с ней будет выводится как «Новость». verbose_name_plural отвечает за множественный вариант названия (например, при работе с несколькими строками).

После того, как модель написана, ее нужно связать с веб-приложением. Это нужно для того, чтобы Django создал конкретную таблицу по написанной модели в базе данных. Для этого используются миграции. С помощью команды makemigration по моделям, для которых еще не существует таблицы в базе данных, создаются таблицы. Затем эти таблицы нужно привязать к текущему веб-приложению с помощью команды migrate. После этого можно использовать созданную таблицу для работы с сайтом.

Также, создаваемые таблицы можно зарегистрировать в Django-приложении admin, которое фреймворк предоставляет для администрирования веб-приложения. Для этого в файле приложения admin.py нужно прописать строчку admin.site.register(name_model), где name_model – имя созданной модели.

2.3 РЕГИСТРАЦИЯ, АВТОРИЗАЦИЯ И РАЗГРАНИЧЕНИЕ ПРАВ ДОСТУПА ПОЛЬЗОВАТЕЛЕЙ ПРИ РАБОТЕ С ВЕБ-ПРИЛОЖЕНИЕМ

Для написания сайта и настраивания базы данных используется фреймворк Django, язык Python, база данных PostgreSQL.

Важно отметить, что сам фреймворк автоматически реализует модель User, отвечающую за хранение информации о пользователях веб-приложения. Эта модель создает в базе данных таблицу user, имеющую 5 атрибутов: username (имя пользователя), password (пароль), email (адрес электронной почты), first_name (настоящее имя пользователя) и last_name (фамилия пользователя). Пароль в базе данных хранится в виде хеша, чтобы его нельзя было извлечь напрямую.

Также фреймворк Django автоматически реализует модель Group, отвечающую за хранение прав пользователей. С ее помощью можно настроить разграничение прав доступа к базе данных на уровне приложения. В нашем веб-приложении существует три роли:

1) Администратор (admin) – пользователь, имеющий полный доступ к базе данных, включая таблицы «user» и «group». Администратор может настраивать права других пользователей. Он возглавляет управление сайтом и включает в себя все права модератора.

2) Модератор (moderator) – пользователь, имеющий доступ к базе данных, но не имеющий доступ к таблицам «user» и «group». Модератор занимается добавлением на сайт новых новостей, фильмов и актеров, а также редактированием комментариев посетителей.

3) Посетитель (visitor) – пользователь, не имеющий доступа к базе данных. Пользователи просматривают сайт и могут оставлять комментарии.

В веб-приложении настроена иерархия модерации комментариев: модераторы не могут удалять комментарии друг друга или администратора, но при этом могут удалять комментарии обычных пользователей и свои. Администратор может удалять любые комментарии, а пользователи – только свои.

2.4 РАЗРАБОТКА И НАСТРОЙКА СКРИПТА ДЛЯ АВТОМАТИЧЕСКОГО БЭКАПА БАЗЫ ДАННЫХ

Для полной автоматизации работы нашего веб-приложения необходимо автоматизировать создание резервных копий его базы данных на случай непредвиденных сбоев в работе сервера. Для этого мы будем использовать приложение фреймворка Django django_cron.

После выгрузки базы данных на локальный сервер нам стоит отправить полученные файлы на какой-нибудь удаленный хост. Для этого будем использовать Яндекс.Диск и библиотеку yaDisk. Для начала нам нужно зарегистрировать новое приложение и получить токен, с помощью которого наше веб-приложение будет соединяться с сервером Яндекс.Диска. Токен будем хранить в переменной YANDEX_TOKEN

```
from os import system
import os
import datetime
import yadisk
import shutil


DATA_BASE = "site"
USERNAME = "postgres"
PASSWORD = "123456789"
HOST = "localhost"
PORT = "5432"
m = os.path.abspath("backup/")
PG_DUMP = m
FILE_NAME = "backup.sql"
ZIP_NAME = f"{str(datetime.date.today())}.zip"
YANDEX_TOKEN = "мой токен"

class doDump(CronJobBase):
    RUN_EVERY_MINS = 60 * 24
    RUN_AT_TIMES = ['2:05']
    schedule = Schedule(run_at_times=RUN_AT_TIMES,
run_every_mins=RUN_EVERY_MINS)  # Задаем график работы автоматической
выгрузки
    code = "main.doDump"  # Код для выполнения

    def do(self):
        system(f'PGPASSWORD={PASSWORD} {PG_DUMP} --dbname={DATA_BASE} --
file={FILE_NAME} --username={USERNAME} --host={HOST} --port={PORT}')

        shutil.make_archive(str(datetime.date.today()), 'zip', 'backup')

        system(f'python manage.py dumpdata > backup/db.json')
        system(f'python manage.py dumpdata auth.user --indent 2 --format xml
> backup/user.xml')

        YANDEX_DIR = "/backups/"

        y = yadisk.YaDisk(token=f"{YANDEX_TOKEN}")
        try:
```

```
        y.mkdir(f"{YANDEX_DIR}")
    except:
        pass
    y.upload(ZIP_NAME, f"{YANDEX_DIR}{ZIP_NAME}")
```

Поле RUN_EVERY_MINS указывает, с каким промежутком в минутах должен вызываться метод do(). RUN_AT_TIMES указывает, начиная с какого системного времени скрипт начнет свою работу. Поле schedule необходимо для автоматизации скрипта, а поле code указывает, метод do() какого класса будет вызывать данный скрипт.

Теперь нужно настроить видимость нашего скрипта для веб-приложения. Для этого в файл settings.py добавим следующий код:

```
CRON_CLASSES = [
    "main.cron.doDump",
]
```

И наконец мы можем запустить работу нашего скрипта с помощью команды терминала «python manage.py runcrons»

2.5 РЕЗУЛЬТАТЫ РАБОТЫ САЙТА

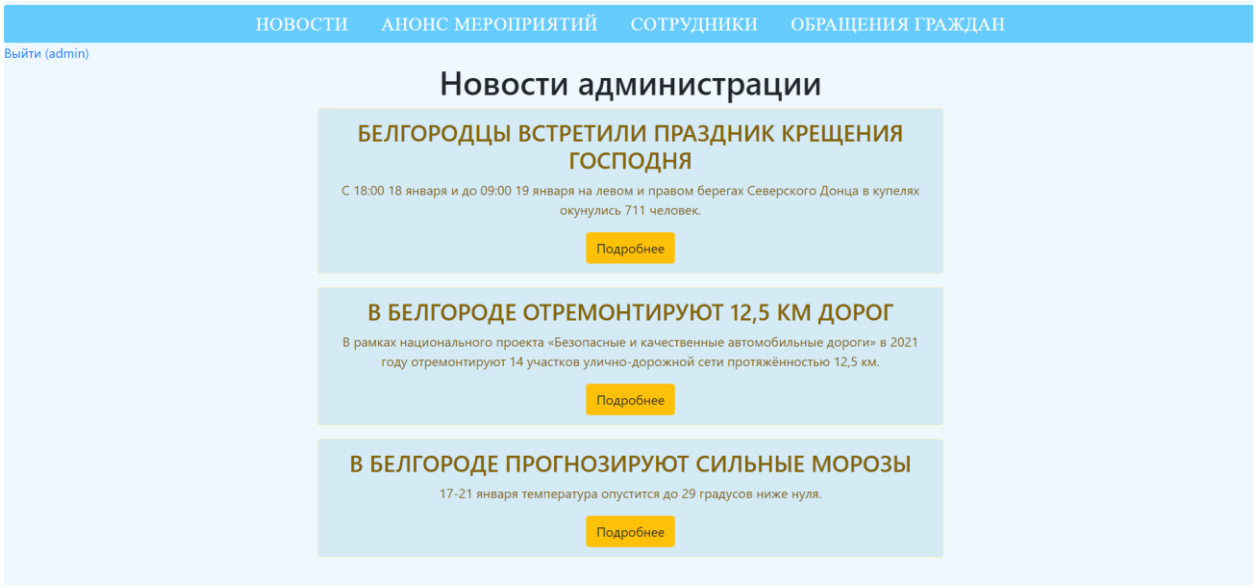


Рис. 1. Страница «Новости»

Рис. 2. Страница «Выбранная новость»

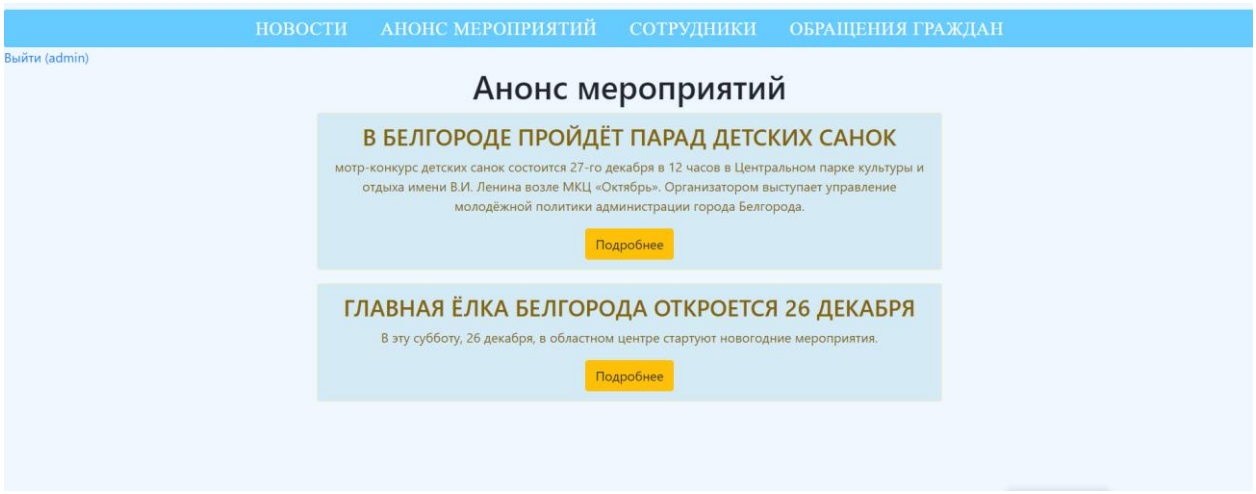


Рис. 3. Страница «Анонс мероприятий»

Рис. 4. Страница «Выбранный анонс мероприятий»

Рис. 5. Страница «Сотрудники»

Рис. 6. Страница «Выбранный Сотрудник»

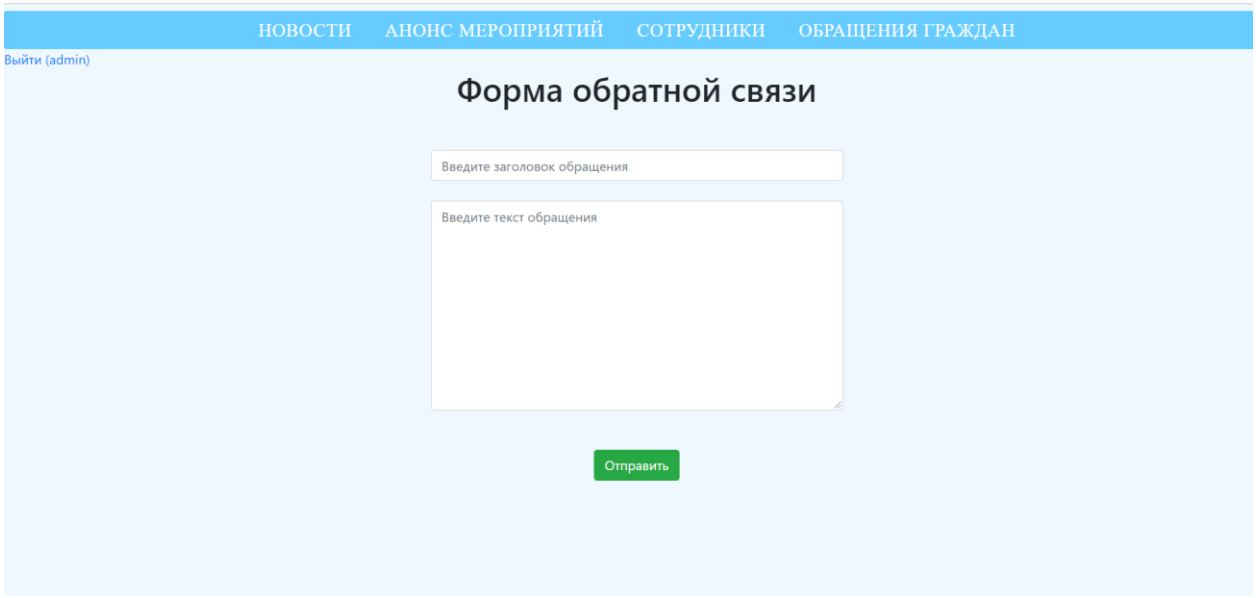


Рис. 7. Страница «Обращения граждан»

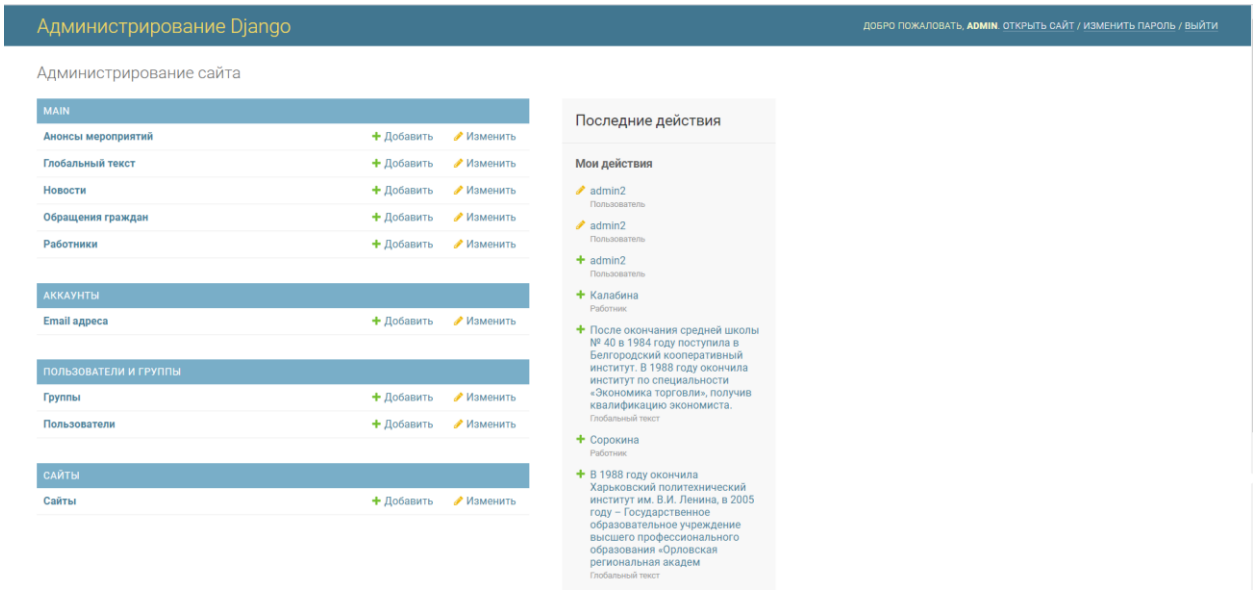


Рис. 8 Администратор admin имеет полный доступ к управлению базой данных

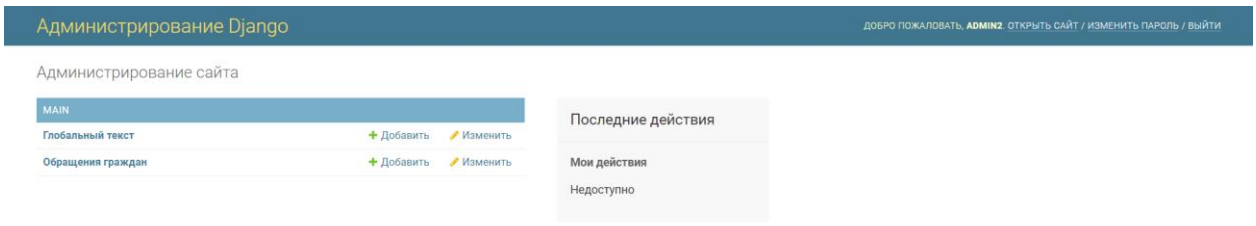


Рис. 9 Модератор admin2 имеет ограниченный доступ к управлению базой данных

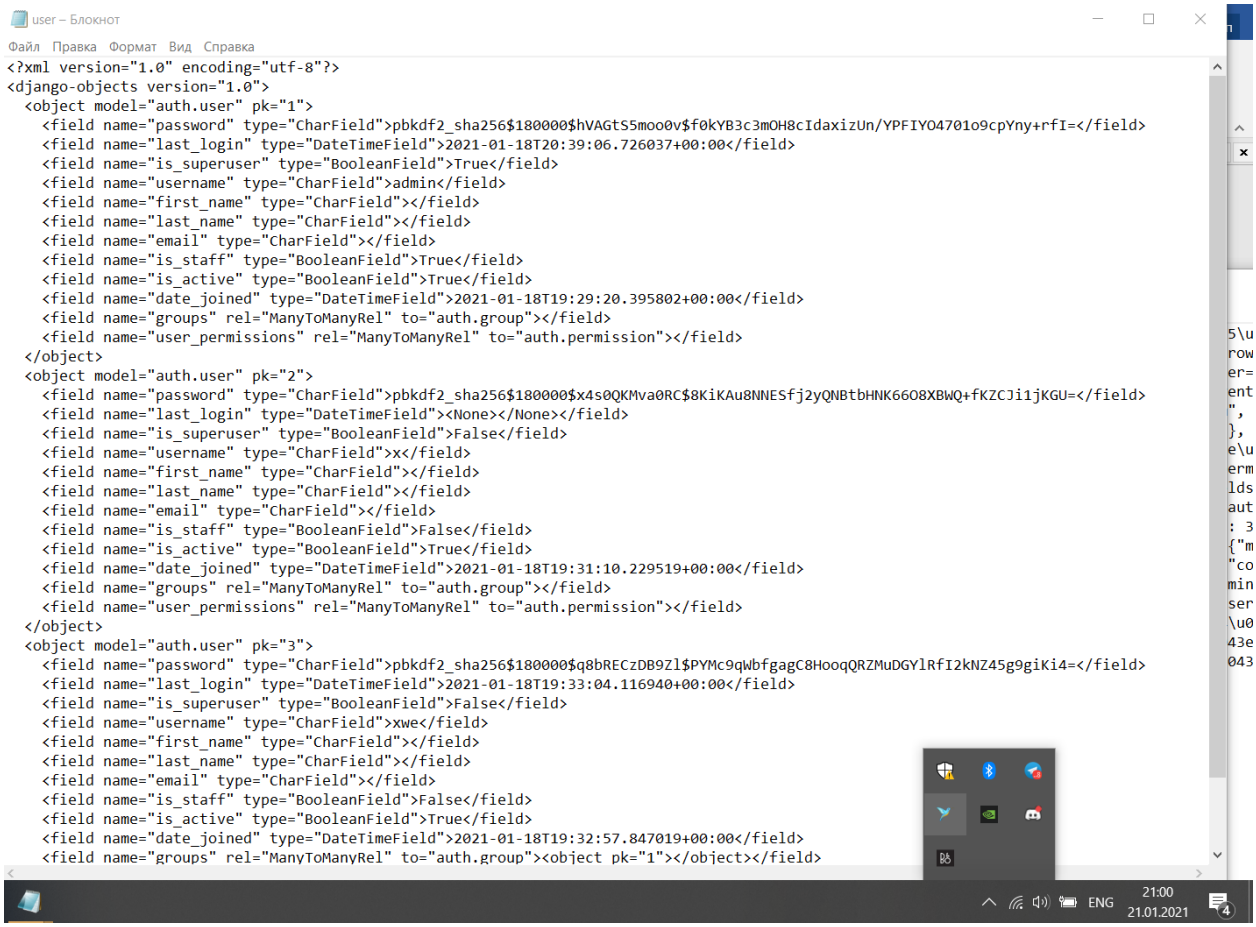


Рис. 10 импорт базы данных в xml

Рис. 11 импорт базы данных в json

Рис. 12 результат бэкапа базы данных на Яндекс.Диск

3. ЗАКЛЮЧЕНИЕ

В процессе написания курсового проекта был получен опыт работы с ORM Django и базой данных PostgrSQL, HTML-разметкой и CSS-стилями, а также с фреймворком Bootstrap

4. СПИСОК ЛИТЕРАТУРЫ

1) https://django.fun/docs/django/ru/3.0/

2) https://postgrespro.ru/docs/postgrespro/

3) http://www.beladm.ru/

4) Bootstrap [Электронный ресурс]. – Электрон. текст. дан. – Режим доступа: https://getbootstrap.com

5. ПРИЛОЖЕНИЕ

5.1 КОД МОДЕЛЕЙ БАЗЫ ДАННЫХ

models.py

```
from django.db import models
from phonenumber_field.modelfields import PhoneNumberField
from django.contrib.auth.models import User


class Text(models.Model):
    text = models.TextField('Текст')

    def __str__(self):
        return self.text[:]

    class Meta:
        verbose_name = 'Глобальный текст'
        verbose_name_plural = 'Глобальный текст'


class News(models.Model):
    title = models.CharField('Название', max_length=50)
    anons = models.CharField('Анонс', max_length=250)
    text = models.ForeignKey(
        Text, related_name='news', on_delete=models.CASCADE, null=False,
verbose_name='Текст'
    )
    date = models.DateTimeField('Дата публикации')

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = 'Новость'
        verbose_name_plural = 'Новости'
        ordering = ['-date']


class Workers(models.Model):
    surname = models.CharField('Фамилия', max_length=20)
    name = models.CharField('Имя', max_length=20)
    middlename = models.CharField('Отчество', max_length=20)
    text = models.ForeignKey(
        Text, related_name='workers', on_delete=models.CASCADE, null=False,
verbose_name='Текст'
    )
    birthdate = models.DateField('Дата рождения')

    def __str__(self):
        return self.surname

    class Meta:
        verbose_name = 'Работник'
        verbose_name_plural = 'Работники'
```

```
class Events(models.Model):
    title = models.CharField('Название', max_length=50)
    anons = models.CharField('Анонс', max_length=250)
    text = models.ForeignKey(
        Text, related_name='events', on_delete=models.CASCADE, null=False,
verbose_name='Текст'
    )
    date = models.DateTimeField('Дата публикации')

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = 'Анонс мероприятия'
        verbose_name_plural = 'Анонсы мероприятий'
        ordering = ['-date']


class Refences(models.Model):
    title = models.CharField('Название', max_length=50)
    text = models.ForeignKey(
        Text, related_name='refences', on_delete=models.CASCADE, null=False,
verbose_name='Текст'
    )
    user = models.ForeignKey(
        User, related_name='user', on_delete=models.CASCADE, null=False,
verbose_name='Индефикатор пользователя'
    )

    def __str__(self):
        return self.title

    class Meta:
        verbose_name = 'Обращения граждан'
        verbose_name_plural = 'Обращения граждан'
```

5.2 КОД ПРЕДСТАВЛЕНИЙ ДЛЯ СВЯЗЫВАНИЯ БД И САЙТА

views.py

```
from django.shortcuts import render
from django.http import HttpResponse
from .models import News, Events, Workers, Refences, Text
from django.views.generic import DetailView
from django.views.generic.base import View
from django.http import HttpResponseRedirect
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm
from django.contrib.auth import logout, login
from django.contrib.auth.models import Group
from django.views.generic import FormView
from .forms import ReferencesForm
# def index (request):
#    return render(request, 'main/index.html')

def index(request):
    news = News.objects.all()
    return render(request, 'main/index.html', {'news': news})


def events(request):
    events = Events.objects.all()
    return render(request, 'main/events.html', {'events': events})


def workers(request):
    workers = Workers.objects.all()
    return render(request, 'main/workers.html', {'workers': workers})


def references(request):
    references = Refences.objects.all()
    return render(request, 'main/references.html', {'references':
references})


class NewsDetailView(DetailView):
    model = News
    template_name = 'main/details_news.html'
    context_object_name = 'news_detail'


class EventsDetailView(DetailView):
    model = Events
    template_name = 'main/details_events.html'
    context_object_name = 'events_detail'

class WorkersDetailView(DetailView):
    model = Workers
    template_name = 'main/details_workers.html'
    context_object_name = 'workers_detail'

class RegisterFormView(FormView): # Форма для регистрации нового пользователя
    form_class = UserCreationForm
    success_url = '../login'

    template_name = 'main/register.html'
```

```
    def form_valid(self, form):
        new_user = form.save()
        new_user.groups.add(Group.objects.get(name='user'))
        new_user.save()
        return super(RegisterFormView, self).form_valid(form)

    def form_invalid(self, form):
        return super(RegisterFormView, self).form_invalid(form)


class LoginFormView(FormView): # Форма для авторизации нового пользователя
    form_class = AuthenticationForm
    success_url = "../"

    template_name = "main/login.html"

    def form_valid(self, form):
        self.user = form.get_user()
        login(self.request, self.user)
        return super(LoginFormView, self).form_valid(form)


class LogoutView(View): # Форма для выхода из аккаунта
    def get(self, request):
        logout(request)
        return HttpResponseRedirect("../")


def create(request):
    error = ''
    if request.method == 'POST':
        form = ReferencesForm(request.POST)
        if form.is_valid():
            text1 = Text(text=form.cleaned_data['text'])
            text1.save()
            ref = Refences(user=request.user,
title=form.cleaned_data['title'], text=text1)
            ref.save()
        else:
            error='Ошибка заполнения формы'
    form = ReferencesForm()
    data = {
        'form': form,
        'error': error
    }
    return render(request, 'main/references.html', data)
```

5.3 HTML-РАЗМЕТКА СТРАНИЦ

layot.html

```
{% load static %}
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
          content="width=device-width, user-scalable=no, initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>{% block title %} {% endblock %}</title>
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.cs
s">
    <link rel="stylesheet" href="{% static 'main/css/main.css' %}">
    <link rel="stylesheet" href="{% static 'main/css/menu.css' %}">
</head>
<body>
<div id="menuDemo">
<!--start CssMenu-->
<div class="menu-icon-wrapper">
   <div class="menu-icon" onclick="toggleCssMenu(this)">
      <div class="three-line">
         <span></span>
         <span></span>
         <span></span>
      </div>
   </div>
</div>
<div id="cssmenu">
    <ul>
        <li><a href="{% url 'news' %}">Новости</a></li>
        <li><a href="{% url 'events' %}">Анонс мероприятий</a></li>
        <li><a href="{% url 'workers' %}">Сотрудники</a></li>
        <li><a href="{% url 'references' %}">Обращения граждан</a></li>
    </ul>
</div>
<tr>
  {% if user.is_authenticated %}
                        <td width="650px" align="right"><span class="GoTo"><a
href="{% url 'logout' %}">Выйти ({{user.username}})</a></span></td>
                    {% else %}
                        <td width="550px" align="right"><span class="GoTo"><a
href="{% url 'register' %}">Регистрация</a></span></td>
                        <td width="110px" align="center"><span
class="GoTo"><a href="{% url 'login' %}">Авторизация</a></span></td>
                    {% endif %}
                </tr>
    <script>
    function toggleCssMenu(icon) {
        var cssmenu = document.getElementById('cssmenu');
        if (icon.className.indexOf('active') == -1) {
            icon.className = 'menu-icon active';
         cssmenu.style.display = "block";
         setTimeout(function(){cssmenu.className = 'active';},0);
        }
        else {
            icon.className = 'menu-icon';
         cssmenu.className = '';
```

```
            setTimeout(function(){cssmenu.style.display = "none";},411);
           }
    }
</script>
    {% block content %} {% endblock %}
</body>
</html>
```

Index.html

```
{% extends 'main/layot.html' %}

{% block title%}
    Новости
{% endblock %}

{% block content%}
    <div class="features" align="center">
         <h1 align="center">Новости администрации</h1>
            {% for el in news %}
                <div class="alert alert-warning">
                    <h3>{{ el.title }}</h3>
                    <p>{{  el.anons}}</p>
                    <a href="{% url 'news-detail' el.id %}" class="btn btn-
warning">Подробнее</a>
                </div>
            {% endfor %}
    </div>
{% endblock %}
```

details_news.html

```
{% extends 'main/layot.html' %}

{% block title%}
   {{ news_detail.title }}
{% endblock %}

{% block content%}
    <div class="features" align="center">

         <h1 align="center">{{ news_detail.title }}</h1>
        <p style="margin-left: 10%; margin-right: 10% " >{{ news_detail.text
}}</p>
         <p>{{ news_detail.date }}</p>
    </div>
{% endblock %}
```

events.html

```
{% extends 'main/layot.html' %}

{% block title%}
    Анонс мероприятий
{% endblock %}

{% block content%}
    <div class="features" align="center">
         <h1 align="center">Анонс мероприятий</h1>
            {% for el in events %}
                <div class="alert alert-warning">
                    <h3>{{ el.title }}</h3>
                    <p>{{  el.anons}}</p>
                    <a href="{% url 'events-detail' el.id %}" class="btn btn-
warning">Подробнее</a>
                </div>
            {% endfor %}
     </div>
{% endblock %}
```

details_events.html

```
{% extends 'main/layot.html' %}

{% block title%}
   {{ events_detail.title }}
{% endblock %}

{% block content%}
    <div class="features" align="center">
         <h1 align="center">{{ events_detail.title }}</h1>
        <p style="margin-left: 10%; margin-right: 10% ">{{ events_detail.text
}}</p>
         <p>{{ events_detail.date }}</p>
    </div>
{% endblock %}
```

workers.html

```
{% extends 'main/layot.html' %}

{% block title%}
    Сотрудники
{% endblock %}

{% block content%}
       <div class="features" align="center">
         <h1 align="center">Работники администрации</h1>
            {% for el in workers %}
                <div class="alert alert-warning">
                    <h3>{{ el.surname }} {{  el.name}}  {{
el.middlename}}</h3>
                <a href="{% url 'workers-detail' el.id %}" class="btn btn-
warning">Подробнее</a>
```

```
                </div>
            {% endfor %}
    </div>
{% endblock %}
```

details_workers.html

```
{% extends 'main/layot.html' %}

{% block title%}
   {{ workers_detail.surname }}
{% endblock %}

{% block content%}
    <div class="features" align="center">

         <h1 align="center">{{ workers_detail.surname }} {{
workers_detail.name }} {{ workers_detail.middlename }}</h1>
         <p style="margin-left: 10%; margin-right: 10% ">{{
workers_detail.text }}</p>
    </div>
{% endblock %}
```

references.html

```
{% extends 'main/layot.html' %}

{% block title%}
    Обращения граждан
{% endblock %}

{% block content%}
    <div class="features" align="center">
         <h1 align="center">Форма обратной связи</h1>
         <form method="post">
        {% csrf_token %}

              {{ form.title }}<br>
              {{ form.text }}<br><br>
             <span>{{error  }}</span>
        <button align="center" class="btn btn-success" type="submit"
>Отправить</button>
    </form>
    </div>
{% endblock %}
```

register.html

```
{% extends 'main/layot.html' %}
{% load static %}
```

```
{% block title %}Регистрация{% endblock %}

{% block content%}
    <h1 align="center">
        Регистрация
    </h1>

    <form action="" method="post">
        {% csrf_token %}
        <div class="forms">
            {{ form.as_p }}
            <button type="submit" style="margin-left: 20vw;" class="btn btn-
warning">Зарегистрироваться</button>
        </div>
    </form>


{% endblock %}
```

login.html

```
{% extends 'main/layot.html' %}
{% load static %}

{% block title %}Авторизация{% endblock %}

{% block content%}
    <h1 align="center">
        Авторизация
    </h1>
    <style>
    .forms {
        margin-top: 5vh;
        margin-left: 30vw;
        width: 40vw;
        float: left;
        color: black;
            }
    </style>
    <form action="" method="post">
        {% csrf_token %}
        <div class="forms">
            {{ form.as_p }}
            <button type="submit" style="margin-left: 10vw;" class="btn btn-
warning">Войти</button>
        </div>
    </form>


{% endblock %}
```

5.4 КОД СКРИПТА ДЛЯ АВТОМАТИЧЕСКОГО БЭКАПА БАЗЫ ДАННЫХ

```
from os import system
import os
import datetime
import yadisk
import shutil


DATA_BASE = "site"
USERNAME = "postgres"
PASSWORD = "123456789"
HOST = "localhost"
PORT = "5432"
m = os.path.abspath("backup/")
PG_DUMP = m
FILE_NAME = "backup.sql"
ZIP_NAME = f"{str(datetime.date.today())}.zip"
YANDEX_TOKEN = "мой токен"

class doDump(CronJobBase):
    RUN_EVERY_MINS = 60 * 24
    RUN_AT_TIMES = ['2:05']
    schedule = Schedule(run_at_times=RUN_AT_TIMES,
run_every_mins=RUN_EVERY_MINS)  # Задаем график работы автоматической
выгрузки
    code = "main.doDump"  # Код для выполнения

    def do(self):
        system(f'PGPASSWORD={PASSWORD} {PG_DUMP} --dbname={DATA_BASE} --
file={FILE_NAME} --username={USERNAME} --host={HOST} --port={PORT}')

        shutil.make_archive(str(datetime.date.today()), 'zip', 'backup')

        system(f'python manage.py dumpdata > backup/db.json')
        system(f'python manage.py dumpdata auth.user --indent 2 --format xml
> backup/user.xml')

        YANDEX_DIR = "/backups/"

        y = yadisk.YaDisk(token=f"{YANDEX_TOKEN}")
        try:
            y.mkdir(f"{YANDEX_DIR}")
        except:
            pass
        y.upload(ZIP_NAME, f"{YANDEX_DIR}{ZIP_NAME}")
```