# **Лабораторная работа № 3**

*Создание первой модели данных и ее регистрация в административном приложении Django*

Логика современных веб-приложений часто требует обращения к базе данных. Такой управляемый данными сайт подключается к серверу базы данных, получает от него данные и отображает их на странице.

Django отлично подходит для создания управляемых данными сайтов, поскольку включает простые и вместе с тем мощные средства для выполнения запросов к базе данных из программы на языке Python.

С помощью команды:

**django-admin.py startproject blog**

**django-admin startproject blog**

создайте проект с названием **blog** в директории **lab3**. Затем, перейдите в папку blog и выполните команду:

**python manage.py startapp articles**

Эта команда создаст в вашем проекте **blog** новое приложение. Выполните необходимую базовую настройку проекта, как это было описано в предыдущих лабораторных работах.

Зайдите в директорию **articles** и в файл **models.py** сохраните код:

from django.db import models

from django.contrib.auth.models import User

class Article(models.Model):

title = models.CharField(max\_length=200)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

text = models.TextField()

created\_date = models.DateField(auto\_now\_add=True)

def \_\_unicode\_\_ (self):

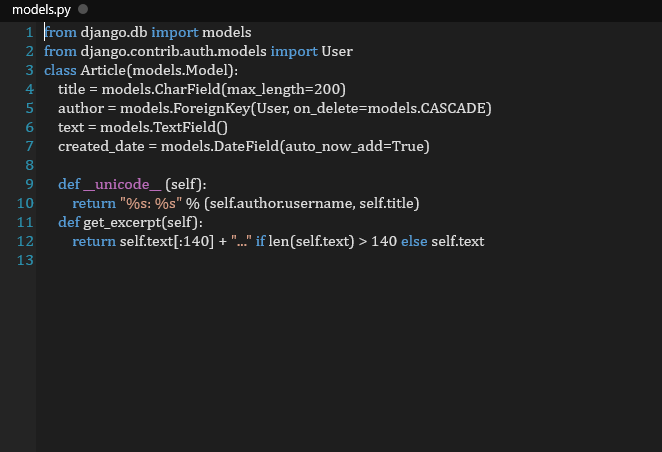
return "%s: %s" % (self.author.username, self.title)

def get\_excerpt(self):

return self.text[:140] + "..." if len(self.text) > 140 else self.text

Будущая модель статей будет иметь 4 поля: заголовок, автор, текст и время создания (в последнем значение будет устанавливаться автоматически).

Метод get\_excerpt позволяет в списке всех статей выводить текст статьи не целиком, а показывать первые 140 символов.



В этой же директории откройте файл **admin.py** (он ответственен за настройку страницы записей в административном приложении. С административным приложением вы ознакомились в ходе выполнения лабораторной работы №1) и сохраните в нем следующий код:

from django.contrib import admin

from .models import Article

class ArticleAdmin(admin.ModelAdmin):

list\_display = ('title', 'author', 'get\_excerpt', 'created\_date')

admin.site.register(Article, ArticleAdmin)

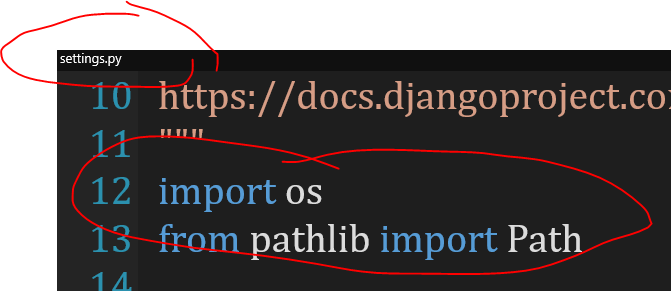
Класс ArticleAdmin нужен для того, чтобы, используя декларативный стиль, описать то, каким образом модель Article должна отображаться в административной панели.

В конце файла вызывается функция admin.site.register(), которой передаются два параметра: модель статей и класс, описывающий, как модель должна отображаться в административном интерфейсе. Эта функция объявляет, что данная модель должна быть добавлена в административный интерфейс.



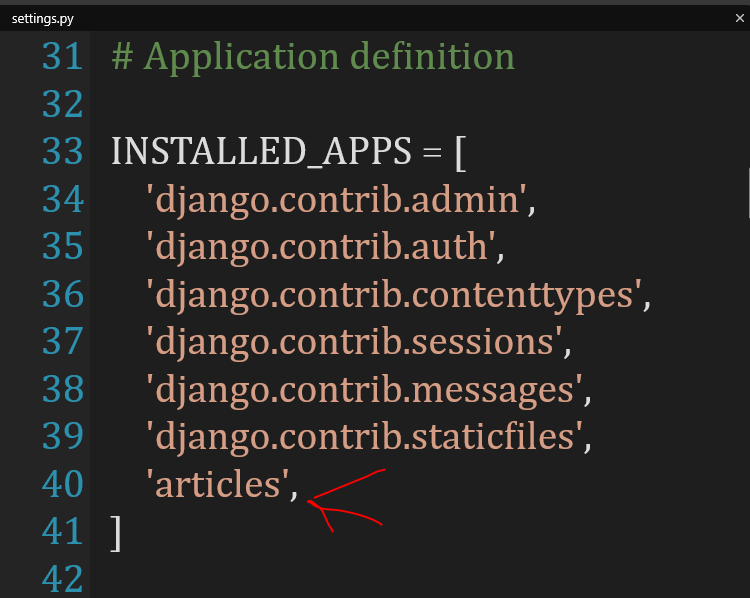
Добавить в файл библиотеку

**import os**



Подключить приложение

в settings.py

**'articles',** 

Создать(изменить строчку name со стандартной на это) в бд

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': os.path.join(BASE\_DIR, 'db\_blog.sqlite3') ,

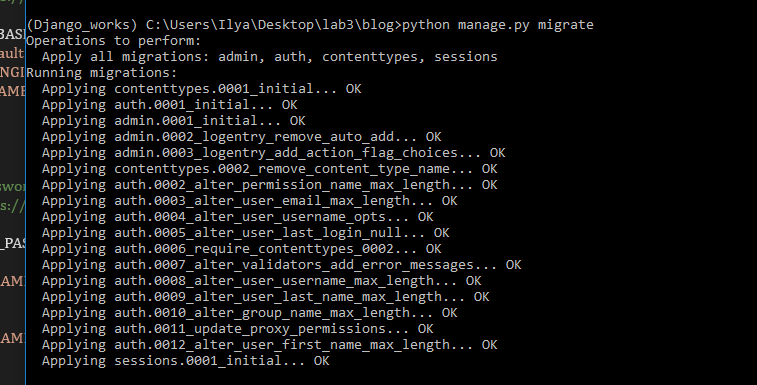
}

}



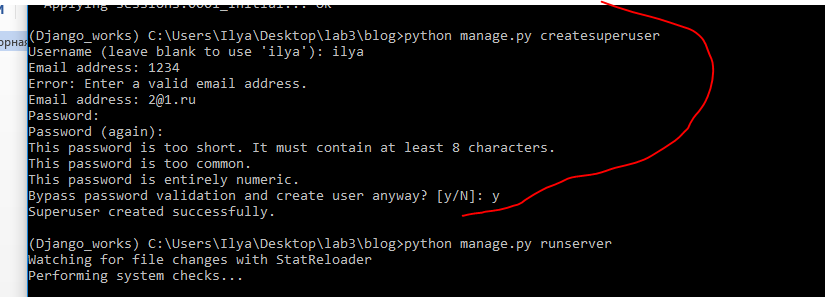
**Сначала мигрируем (т.е. создаем бд)**

**python manage.py migrate**



**Потом создаём супер пользователя (администратор)**

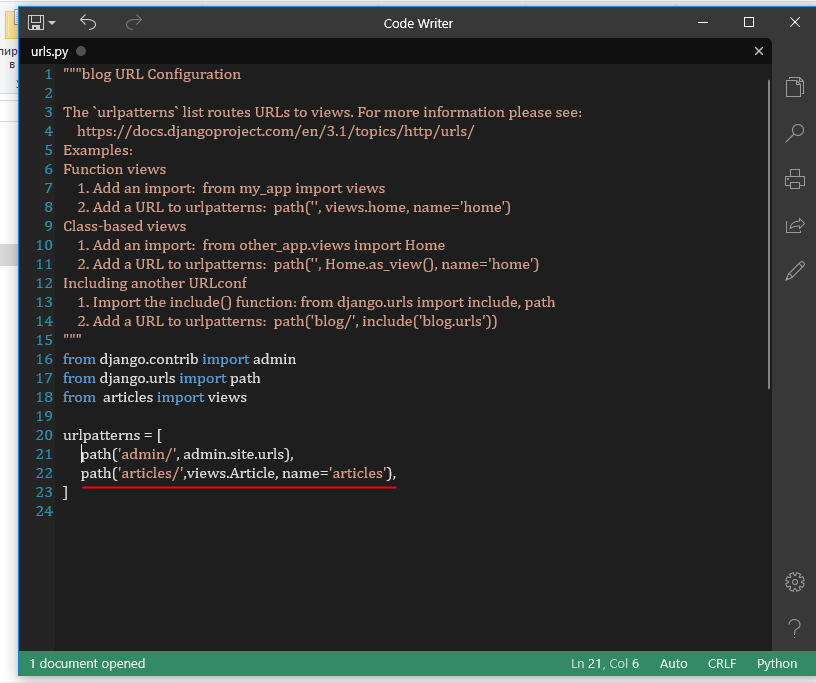
**python manage.py createsuperuser**



**запускаем сервер, проверяем работает ли всё это .**

**python manage.py runserver**

**Добавил в файл (lab3\blog\blog\urls.py)**



from django.contrib import admin

from django.urls import path

from articles import views

urlpatterns = [

path('admin/', admin.site.urls),

path('articles/',views.Article, name='articles'),

]

**файл(lab3\blog\articles\admin.py)**

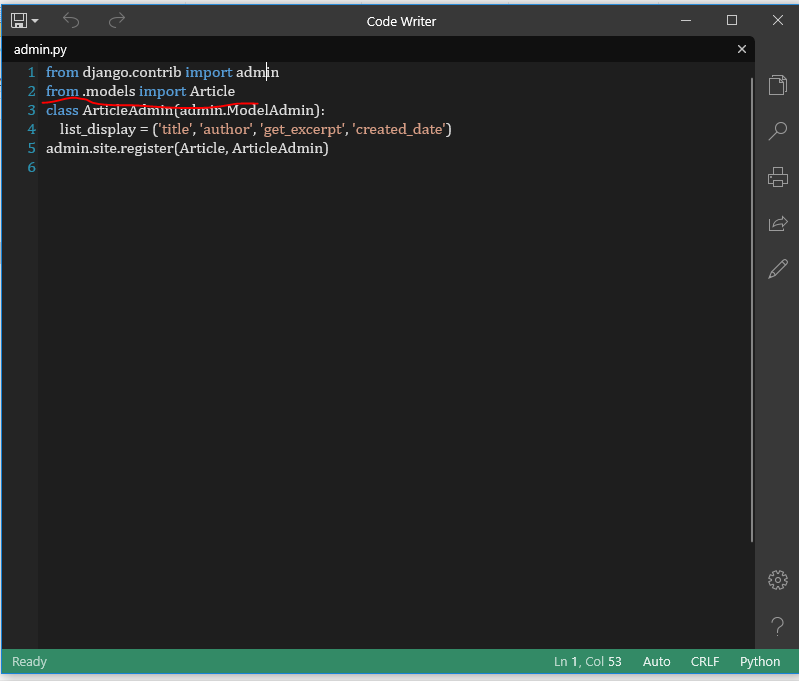
from django.contrib import admin

from .models import Article

class ArticleAdmin(admin.ModelAdmin):

list\_display = ('title', 'author', 'get\_excerpt', 'created\_date')

admin.site.register(Article, ArticleAdmin)



Файл модели(\lab3\blog\articles\models.py)

from django.db import models

from django.contrib.auth.models import User

class Article(models.Model):

title = models.CharField(max\_length=200)

author = models.ForeignKey(User, on\_delete=models.CASCADE)

text = models.TextField()

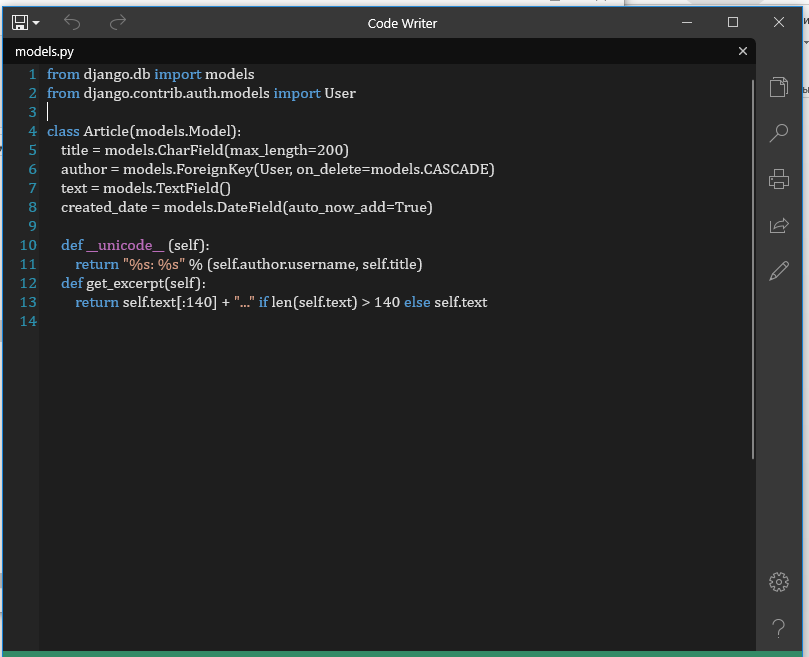
created\_date = models.DateField(auto\_now\_add=True)

def \_\_unicode\_\_ (self):

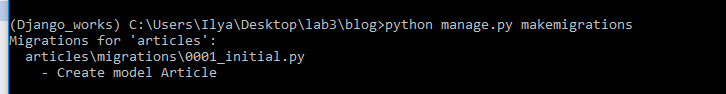
return "%s: %s" % (self.author.username, self.title)

def get\_excerpt(self):

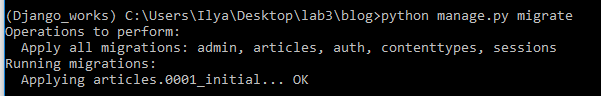
return self.text[:140] + "..." if len(self.text) > 140 else self.text



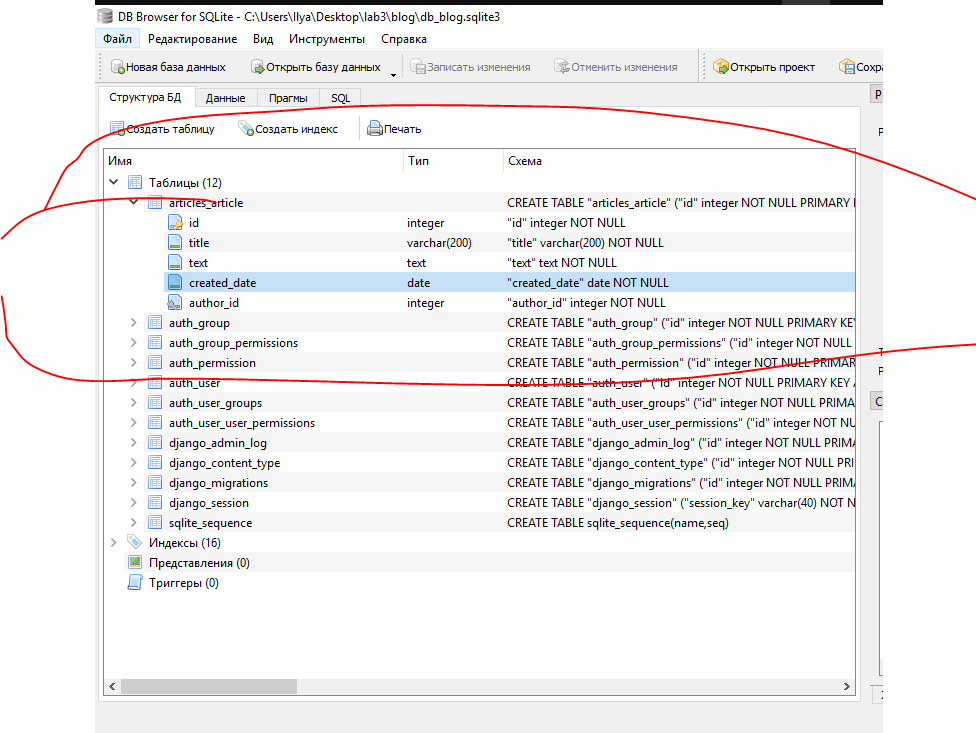
python manage.py makemigrations создаёт модель бд

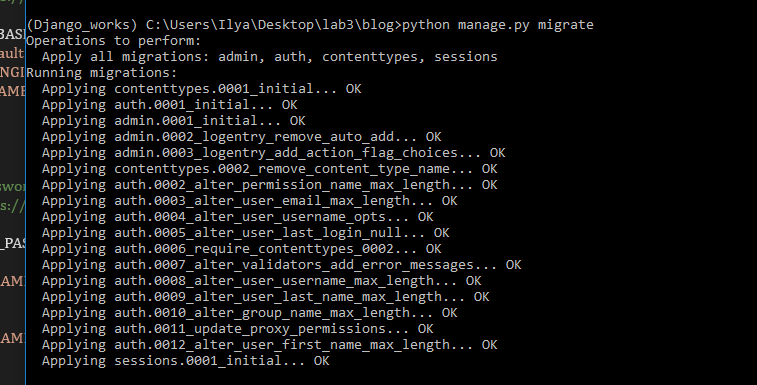


**python manage.py migrate**



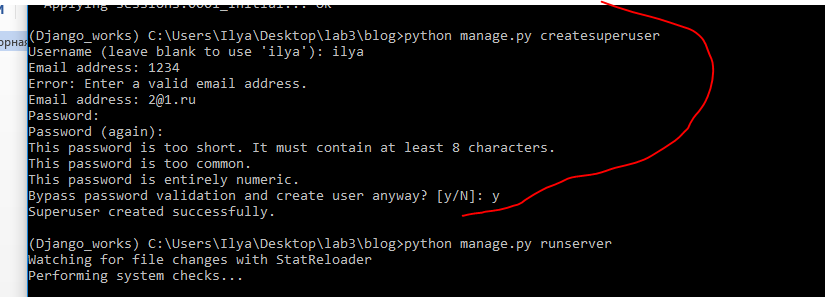
**Проверяем файл создалась ли бд с помощь программы (db browser for sqlite)**





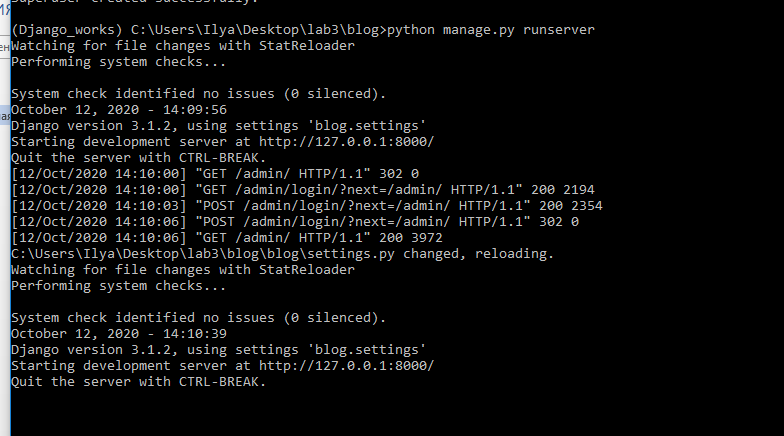
Создаём супер пользователя

**python manage.py createsuperuser**



Теперь нужно подрубить сервер

**python manage.py runserver**



Для проверки правильности выполнения задания откройте административную панель по адресу: <http://127.0.0.1:8000/admin/>. Она должна выглядеть так:

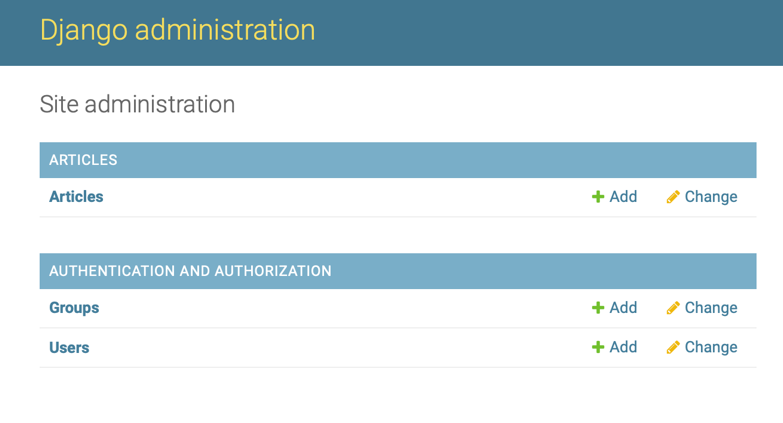
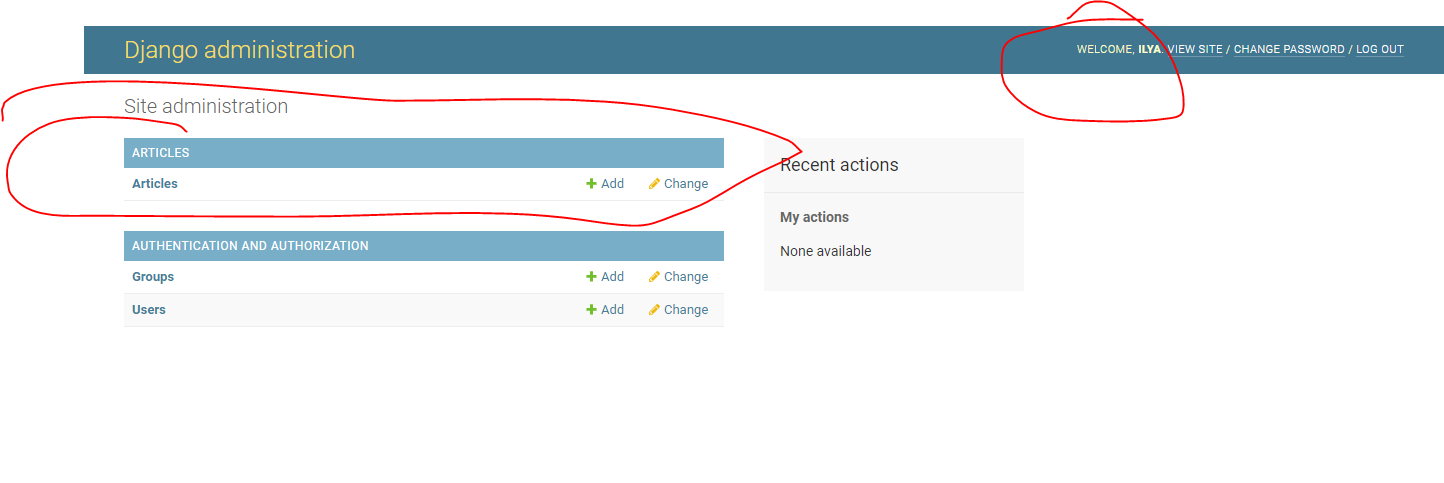
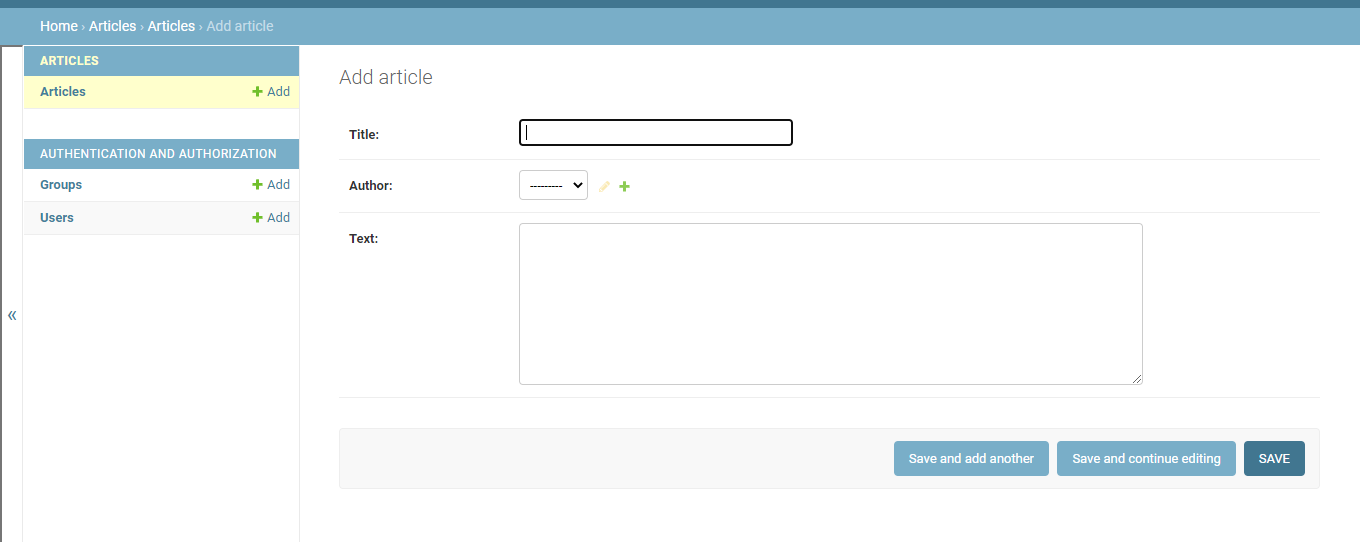


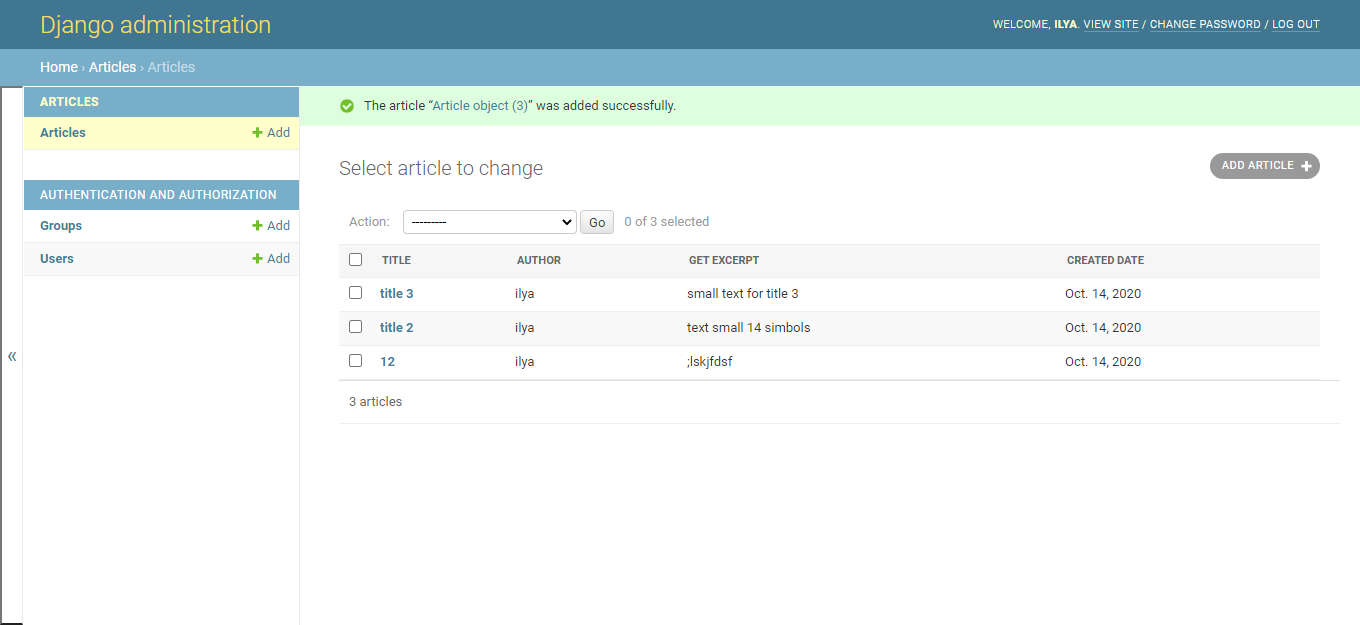
Рисунок 4. Административная панель

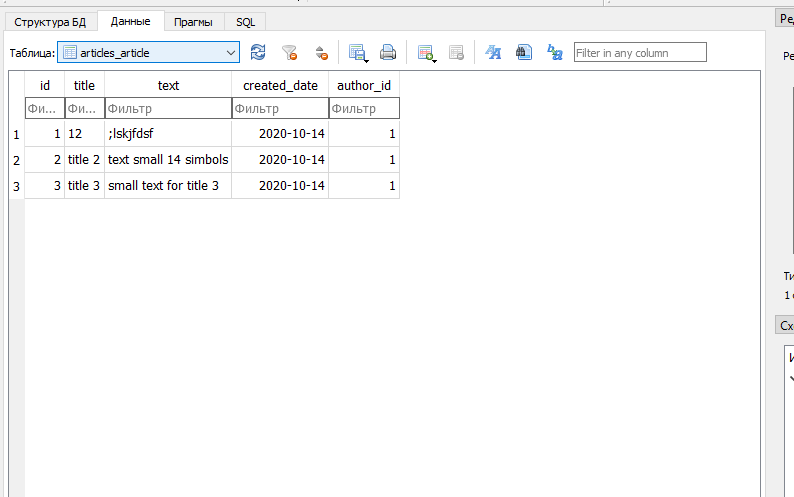


**Добавление данных**

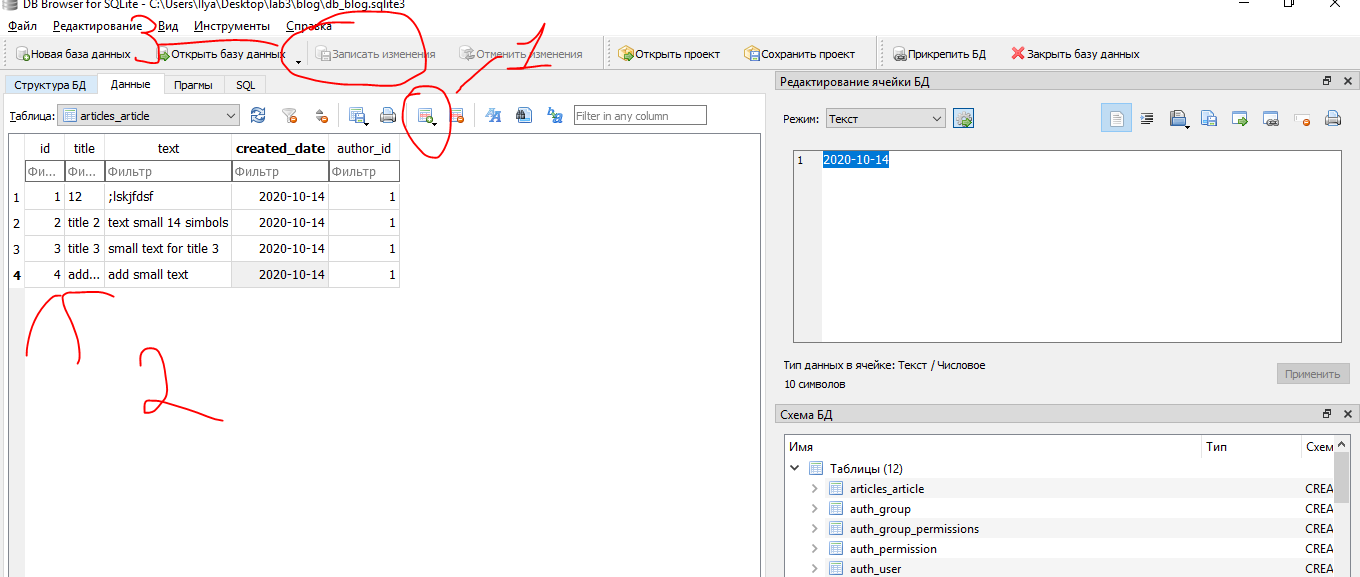


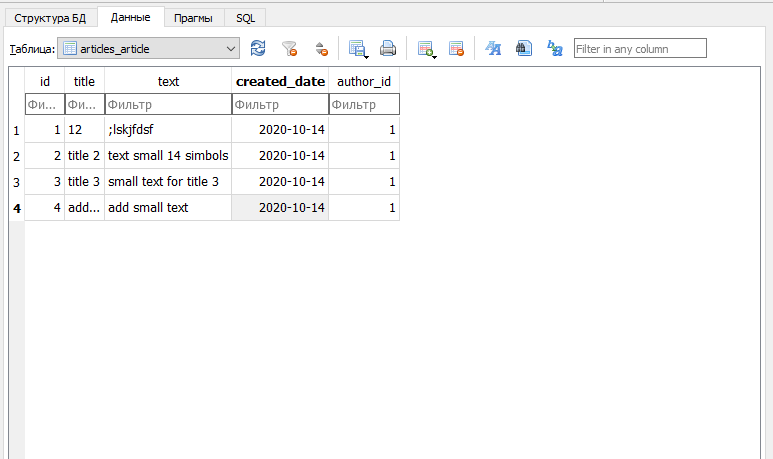
**Задание**:

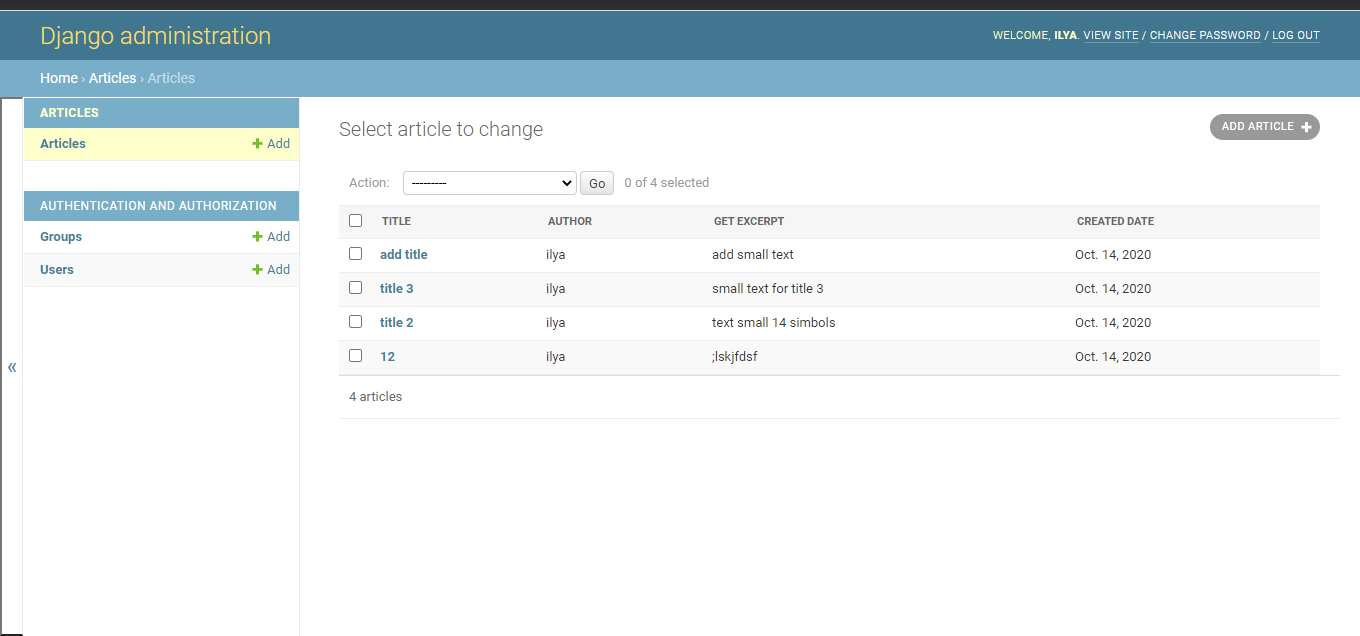
* Перейдите во вкладку Articles и создайте 3 статьи, заполнив все поля. 
* С помощью программы управления базами данных sqlite3 (например, SQLite Manager)

 откройте файл вашей базы данных текущего проекта, который хранится в папке проекта с именем, объявленным в настройках проектах в переменной «**DATABASES.NAME**». Найдите созданные в предыдущем пункте задания экземпляры записей. Измените текст одной записи и название статьи для другой. Создайте еще одну статью.

*Динамическое генерирование шаблона для вывода всех экземпляров этой модели*







В директории **articles** создайте папку **templates**, внутри которой создайте файл **archive.html**. В созданном файле шаблона в качестве названия страницы (тег <title>) укажите фразу «Архив статей». Затем в тег <body> добавьте 2 тега <div>:

<body>

<div class="header">

</div>

<div class="archive">

</div>

</body>

У первого тега <div> установлен класс header, а у второго – archive. Это помогает делать верстку понятной, потому что классы играют роль имен для каждого элемента, из названия класса становится понятно, для чего существует текущий элемент, а также помогает отличать нужные блоки друг от друга при установке стилей.

Во внутрь тега div с классом header добавьте изображение (например, логотип проекта). Пример добавления:

Во внутрь тега div с классом archive добавьте шаблон для отображения одной статьи. Так как у каждой записи есть название, автор, текст и время создания, для каждого поля записи нужно создать определенный элемнт в разметке страницы. Шаблон одного поста будет выглядеть так:

<div class="one-post">

<h2 class="post-title">{{ post.title }}</h2>

<div class="article-info">

<div class="article-author">{{ post.author.username }}</div>

<div class="article-created-date">{{ post.created\_date }}</div>

</div>

<p class="article-text">{{ post.get\_excerpt }}</p>

</div>

Поле «название» было помещено в тег <h2> c классом post-title, поля «автор» и «время создания» были помещены в один общий тег <div>, потому что в будущем эти два поля будут визуально находится на одной строке. Поле «текст» обрамлено тегом <p>, который означает один абзац текста (p – сокращение от «paragraph»). Ограничение на отображение только части текста статьи введено из-за того, что некоторые посты могут по размерам занимать несколько страниц, что, недопустимо при отображении списка сразу многих экземпляров.

Однако созданная разметка подходит для отображения одной статьи, а не нескольких. Для корректного отображения необходимо добавить цикл, который бы повторялся столько раз, сколько статей передано в контекст шаблона. Для этого существует шаблоный тег {% for item in list %}, практически идентичен циклу for языка Python. Переменная posts, передаваемая в шаблон, выполняет роль массива. Для того, чтобы разметка поддерживала отображение сразу многих записей, достаточно добавить строку начала цикла и строку его завершения:

{% for post in posts %}

<div class="one-post">

<h2 class="post-title">{{ post.title }}</h2>

<div class="article-info">

<div class="article-author">{{ post.author.username }}</div>

<div class="article-created- date">{{ post.created\_date }}</div>

</div>

<p class="article-text">{{ post.get\_excerpt }}</p>

</div>

{% endfor %}

Теперь шаблон готов.

В файле **views.py** в директории **articles** создайте представление **archive**, которое будет возвращать html-страницу со всеми созданными постами в текущем проекте. Код представления:

from .models import Article

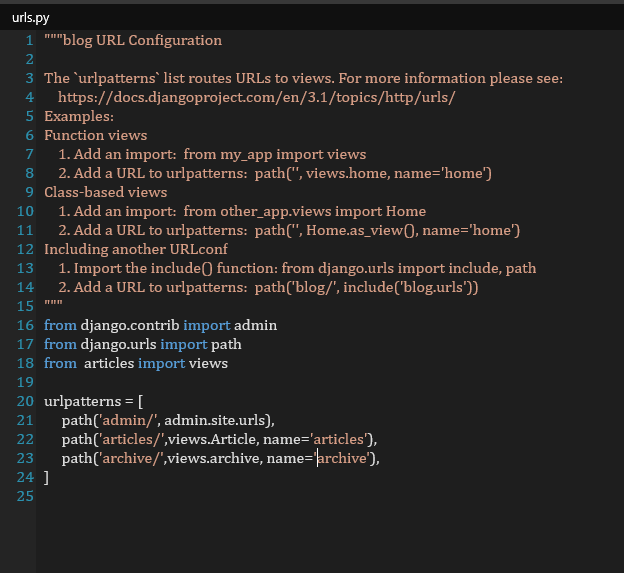
from django.shortcuts import render

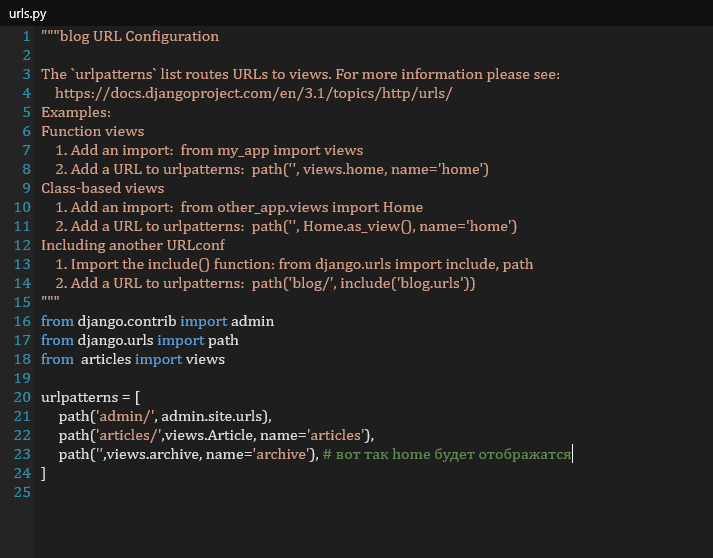
def archive(request):

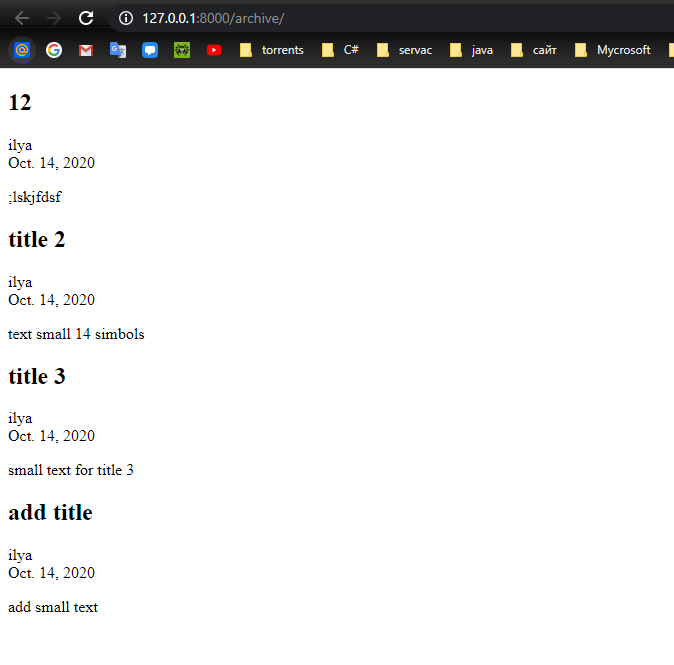
return render(request, 'archive.html', {"posts": Article.objects.all()})



Теперь необходимо настроить url, по которому будут отображаться все статьи проекта. Выполните это задание самостоятельно, указав Django, что при заходе пользователя на главную страницу нужно отображать список всех записей.







**Задание:**

* Откройте файл базы данных, где хранятся экземпляры статей текущего проекта, с помощью программы управления базами данных sqlite3 и добавьте новую запись в блог через менеджер базы;
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).