**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ**

**КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Ордена трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

**Кафедра «Сетевые информационные технологии и сервисы»**



**Отчет по лабораторным работам**

по предмету «Прикладные информационные системы»

Выполнил студент 3 курса

Группа: УБСТ2202

ФИО: Старостин Данил Александрович

**Москва, 2025**

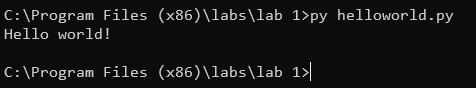
Лабораторная работа № 1.

Начало работы с Python и Django

Цель работы: освоение базовых навыков работы с языком программирования Python и фреймворком Django, включая установку среды разработки, создание простейших скриптов и первого Django-проекта.

Ход выполнения

helloworld.py, содержащий команду вывода строки "Hello world!";



mygroup.py, реализующий структуру данных в виде списка словарей, описывающих студентов группы с полями: имя, фамилия, экзамены, оценки.

# Создание списка словарей для хранения данных о студентах.

# Список (list) - это упорядоченная, изменяемая коллекция объектов. Создается с помощью [].

# Словарь (dict) - это неупорядоченная коллекция пар "ключ: значение". Создается с помощью {}.

groupmates = [

    {

        "name": "Александр",

        "surname": "Иванов",

        "exams": ["Информатика", "ЭЭиС", "Web"],

        "marks": [4, 3, 5]

    },

    {

        "name": "Иван",

        "surname": "Петров",

        "exams": ["История", "АиГ", "КТП"],

        "marks": [4, 4, 4]

    },

    {

        "name": "Кирилл",

        "surname": "Смирнов",

        "exams": ["Философия", "ИС", "КТП"],

        "marks": [5, 5, 5]

    },

    {

        "name": "Мария",

        "surname": "Сидорова",

        "exams": ["Информатика", "ЭЭиС", "Web"],

        "marks": [5, 5, 4]

    },

    {

        "name": "Дарья",

        "surname": "Кузнецова",

        "exams": ["История", "АиГ", "КТП"],

        "marks": [3, 4, 3]

    }

]

# def - ключевое слово для объявления (создания) функции.

# print\_students - имя функции.

# (students) - параметр, который функция принимает. В данном случае, это список студентов.

def print\_students(students):

    """

    Функция для красивого вывода списка студентов в виде таблицы.

    """

    # Выводим заголовок таблицы.

    # u"Имя" - префикс u (от Unicode) использовался в старых версиях Python 2 для обозначения Unicode-строк.

    # В Python 3 все строки по умолчанию являются Unicode, поэтому этот префикс не обязателен, но для совместимости его можно оставить.

    # .ljust(15) - это метод строки, который выравнивает ее по левому краю, добавляя справа пробелы до общей длины в 15 символов.

    # Аналогичная реализация: можно было бы вручную добавлять пробелы, например: "Имя" + ' ' \* (15 - len("Имя"))

    print("Имя".ljust(15), "Фамилия".ljust(10), "Экзамены".ljust(30), "Оценки".ljust(20))

    # Начинаем цикл for, который перебирает каждый элемент (студента) в списке students.

    # student - это временная переменная, которая на каждой итерации цикла содержит один словарь из списка.

    # Аналогичная реализация: можно использовать цикл while:

    # i = 0

    # while i < len(students):

    #   student = students[i]

    #   ... (код внутри цикла) ...

    #   i += 1

    for student in students:

        # Извлекаем значения из словаря по ключам, например, student["name"].

        # str(student["exams"]) - преобразуем список экзаменов в строку для вывода.

        print(student["name"].ljust(15), student["surname"].ljust(10), str(student["exams"]).ljust(30), str(student["marks"]).ljust(20))

def filter\_students\_by\_avg\_mark(students):

    """

    Функция для фильтрации студентов по среднему баллу, который вводится с клавиатуры.

    """

    # input() - функция для получения данных от пользователя из консоли. Она всегда возвращает строку.

    # float() - функция для преобразования строки в число с плавающей точкой (дробное).

    # Это нужно, чтобы можно было сравнивать средний балл с введенным значением.

    # Аналогичная реализация: можно было бы сначала получить строку, а потом преобразовать ее в отдельной строке:

    # mark\_str = input("Введите средний балл для фильтрации: ")

    # avg\_mark\_threshold = float(mark\_str)

    try:

        avg\_mark\_threshold = float(input("Введите пороговый средний балл для фильтрации: "))

    except ValueError:

        print("Ошибка: введено не число. Пожалуйста, перезапустите и введите корректное число.")

        return

    # Создаем пустой список, в который будем добавлять студентов, прошедших фильтрацию.

    # Аналогичная реализация: filtered\_students = list()

    filtered\_students = []

    # Перебираем каждого студента в исходном списке.

    for student in students:

        # sum(student["marks"]) - функция sum() суммирует все числовые элементы в списке.

        # len(student["marks"]) - функция len() возвращает количество элементов в списке.

        # Вычисляем средний балл для текущего студента.

        # Аналогичная реализация: можно посчитать сумму и количество вручную в цикле:

        # total\_marks = 0

        # for mark in student["marks"]:

        #   total\_marks += mark

        # avg\_mark = total\_marks / len(student["marks"])

        avg\_mark = sum(student["marks"]) / len(student["marks"])

        # if - условный оператор. Проверяем, что средний балл студента выше порога, введенного пользователем.

        if avg\_mark > avg\_mark\_threshold:

            # .append() - метод списка, который добавляет элемент в конец списка.

            # Аналогичная реализация: filtered\_students = filtered\_students + [student] (менее эффективный способ)

            filtered\_students.append(student)

    # Проверяем, нашлись ли студенты.

    if filtered\_students:

        print(f"\\nСтуденты со средним баллом выше {avg\_mark\_threshold}:")

        # Вызываем функцию print\_students, чтобы вывести отфильтрованный список.

        print\_students(filtered\_students)

    else:

        print(f"\\nНет студентов со средним баллом выше {avg\_mark\_threshold}.")

# Сначала выводим список всех студентов.

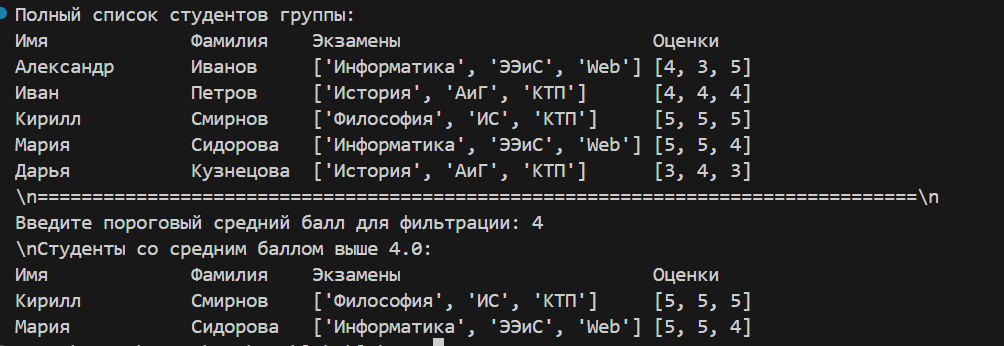
print("Полный список студентов группы:")

print\_students(groupmates)

print("\\n" + "="\*80 + "\\n") # Печатаем разделитель для наглядности

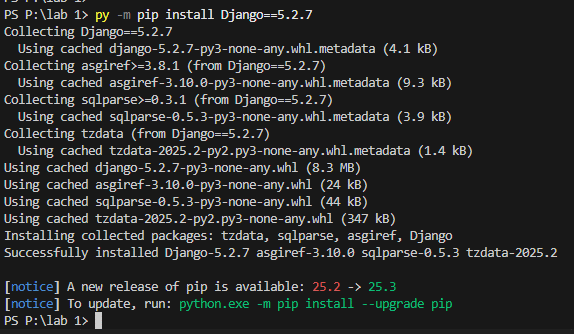
# Затем запускаем функцию фильтрации.

filter\_students\_by\_avg\_mark(groupmates)



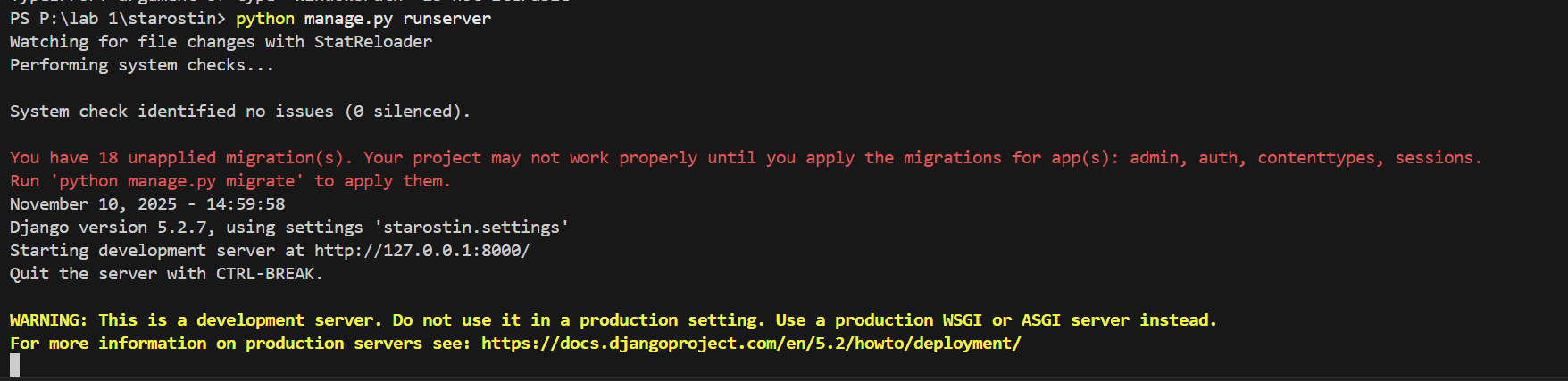
Начало работы с Django

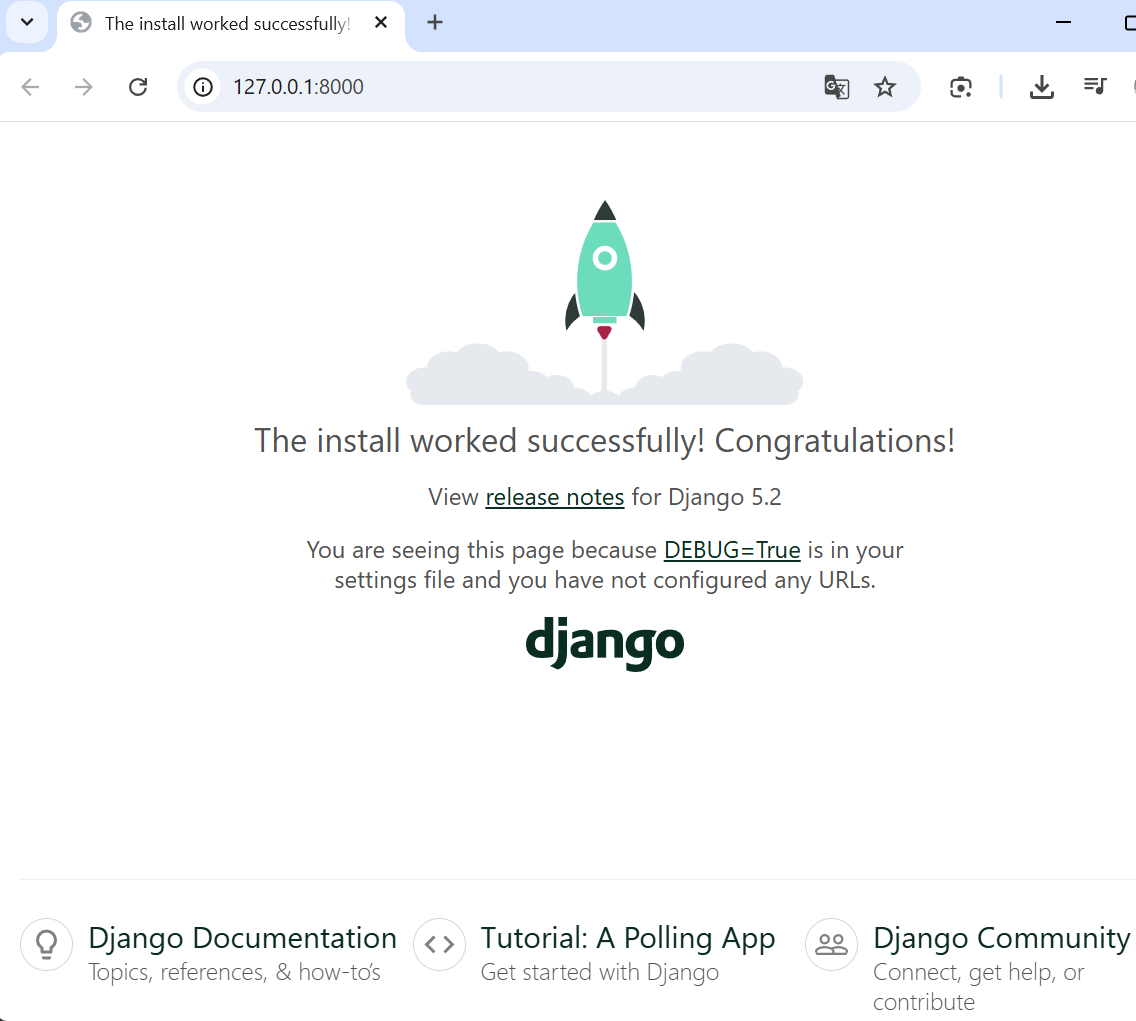
Устанавливаем последнюю версию Django согласно рекомендации с официального сайта



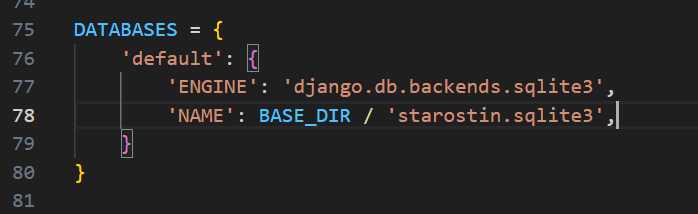
Создаю проект с помощью команды django-admin startproject,

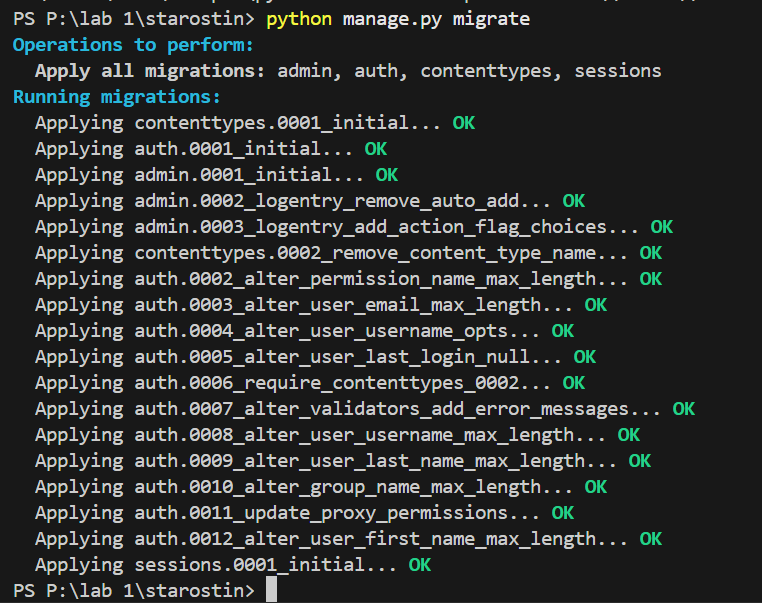


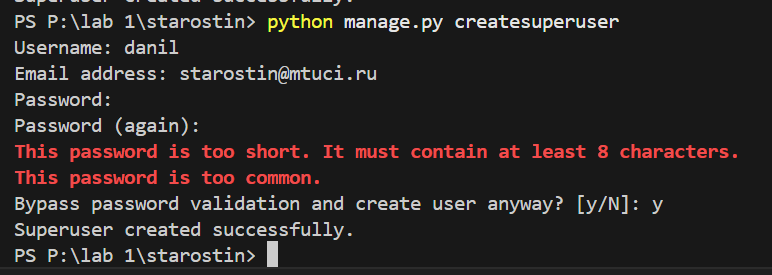


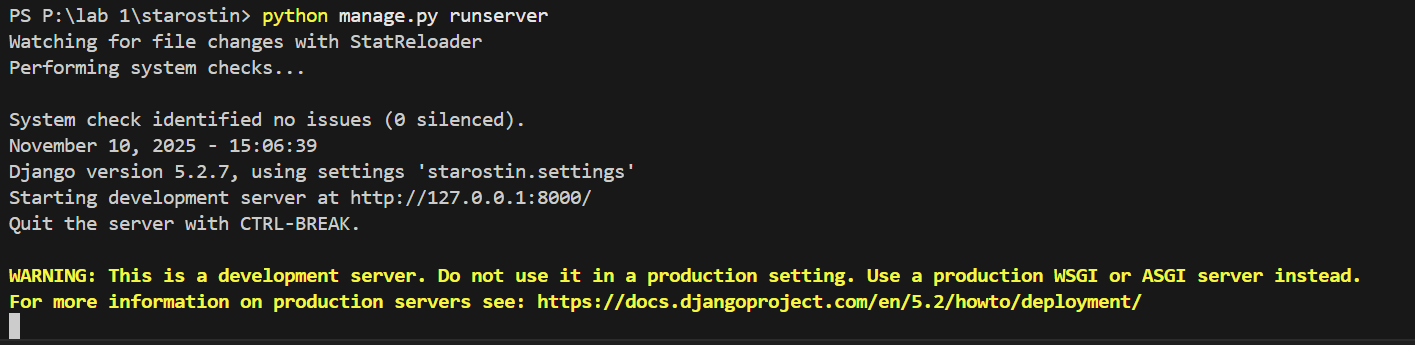


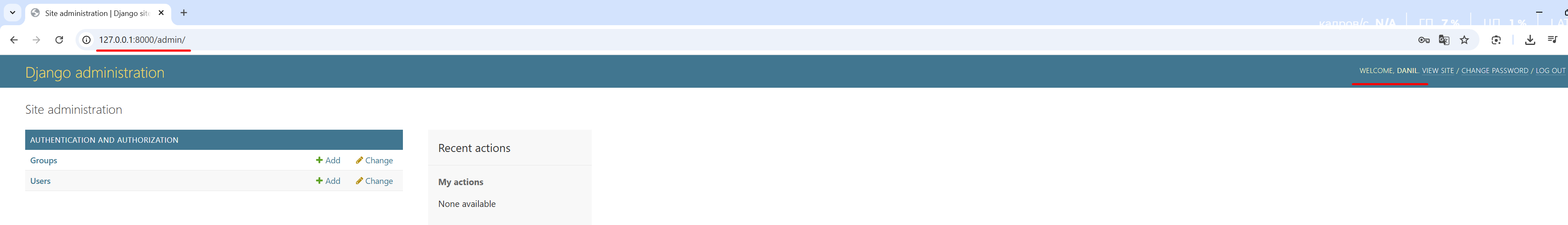
произведена первичная настройка базы данных

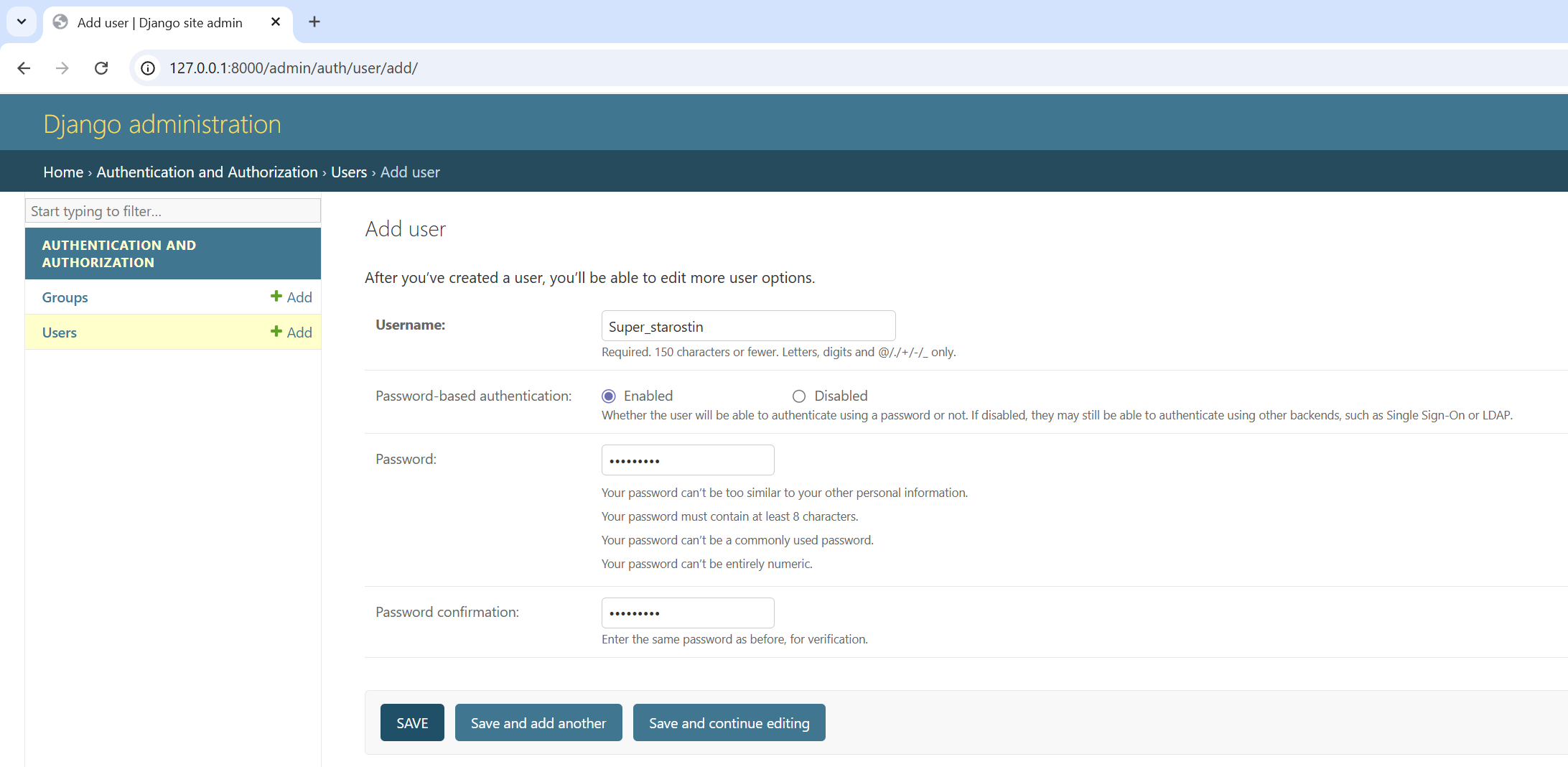
 через выполнение миграций (migrate)

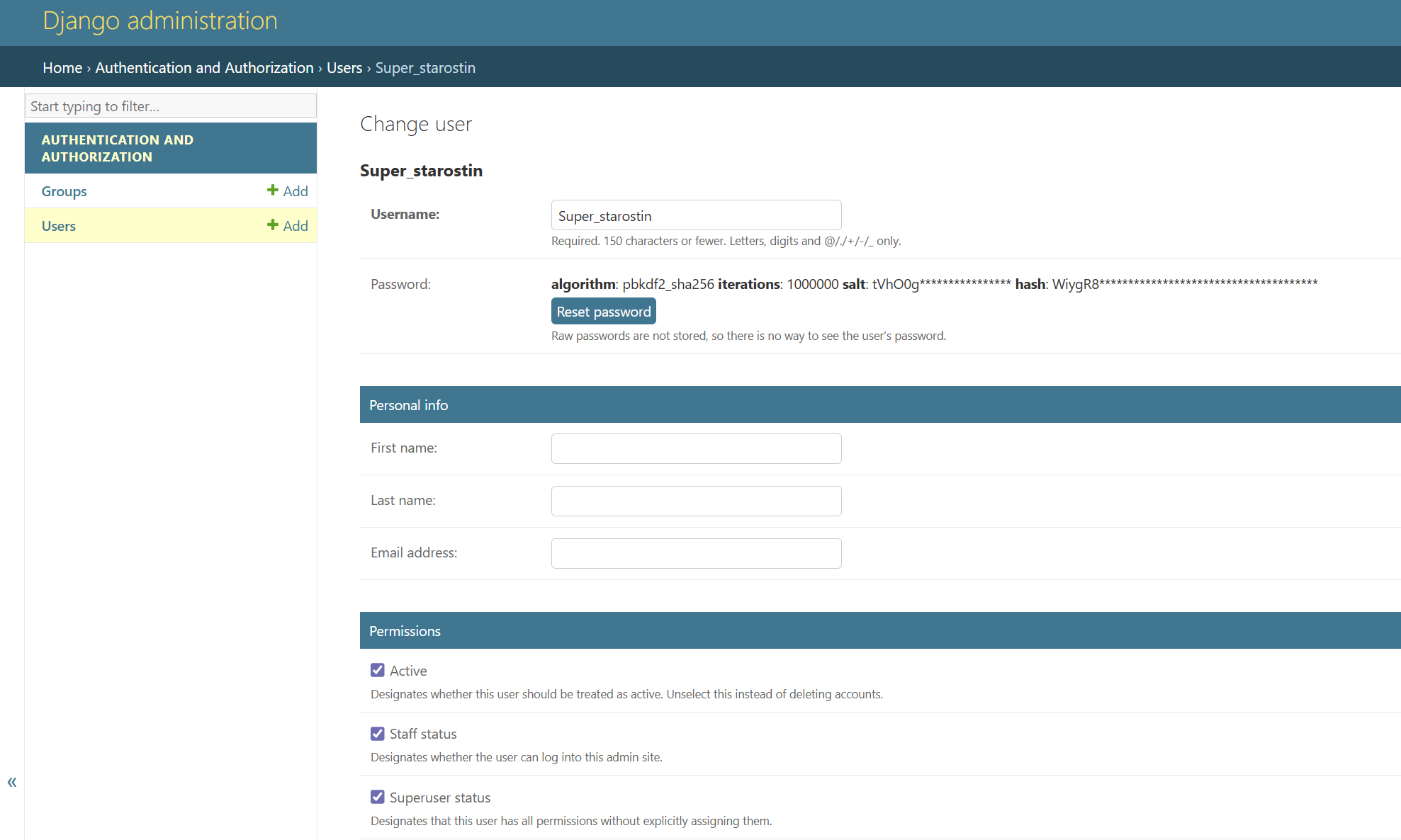
 и создан суперпользователь (createsuperuser).

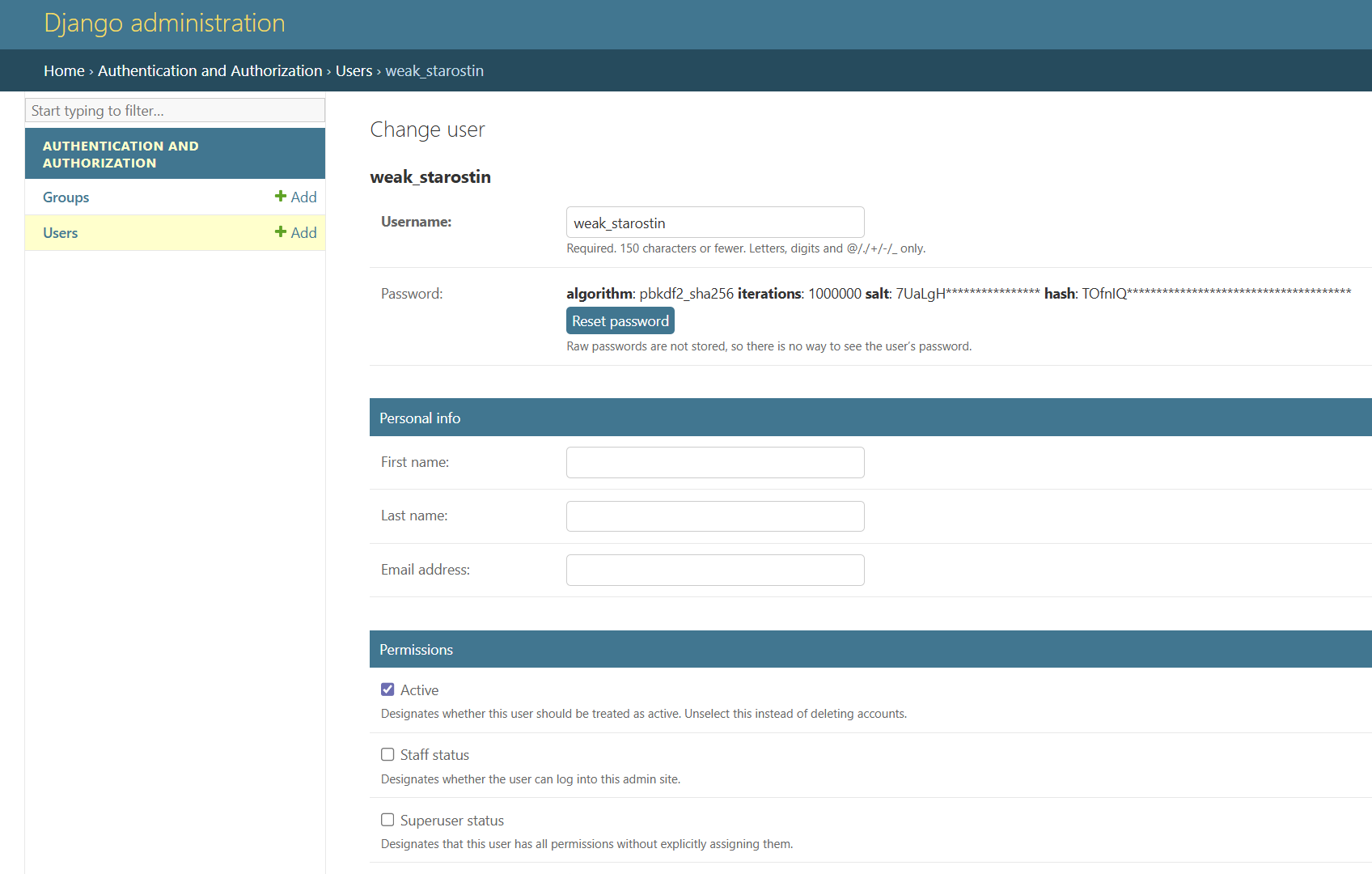
 Выполнен запуск встроенного веб-сервера разработки

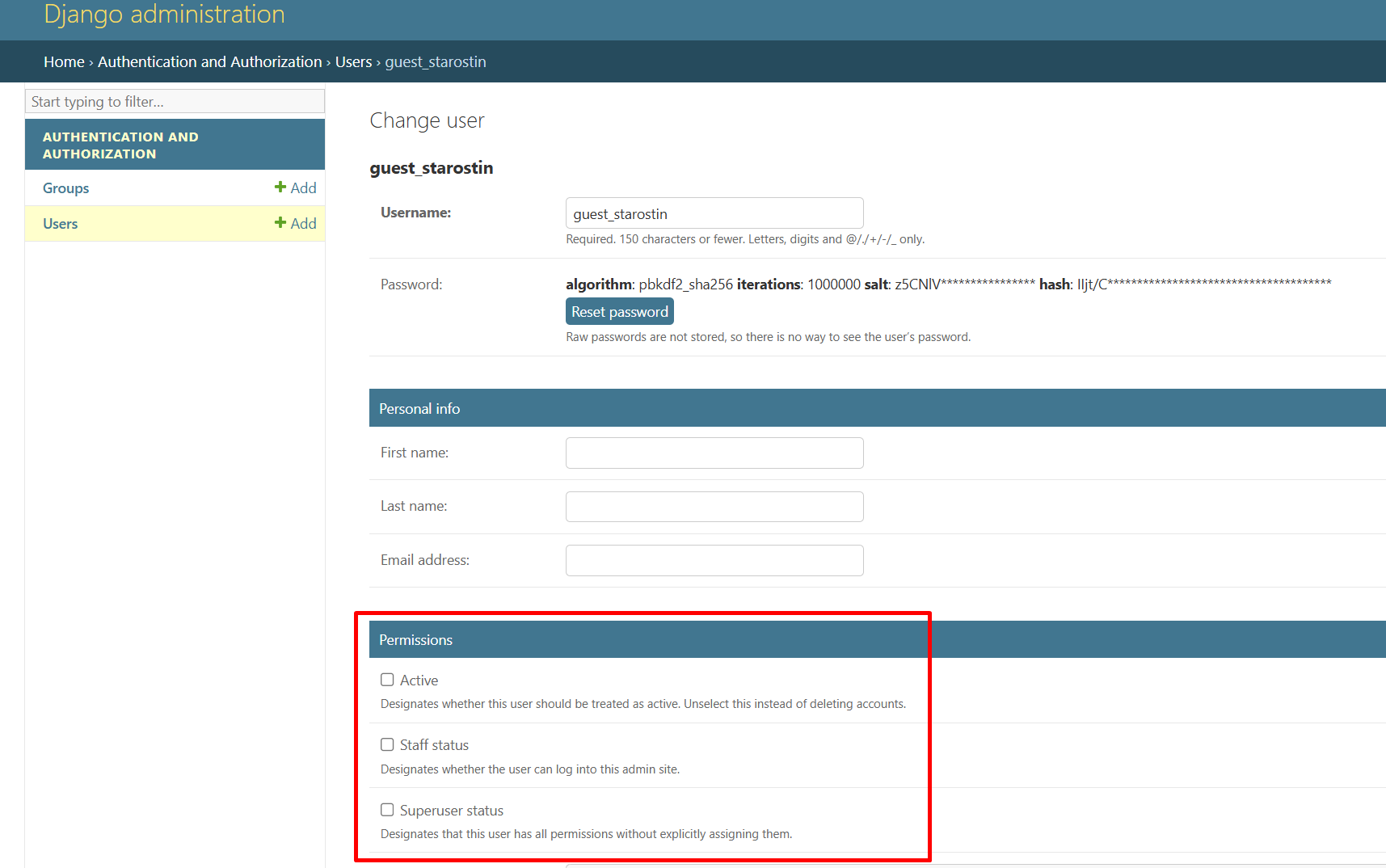
 и проверка работоспособности проекта по адресу http://127.0.0.1:8000/ .

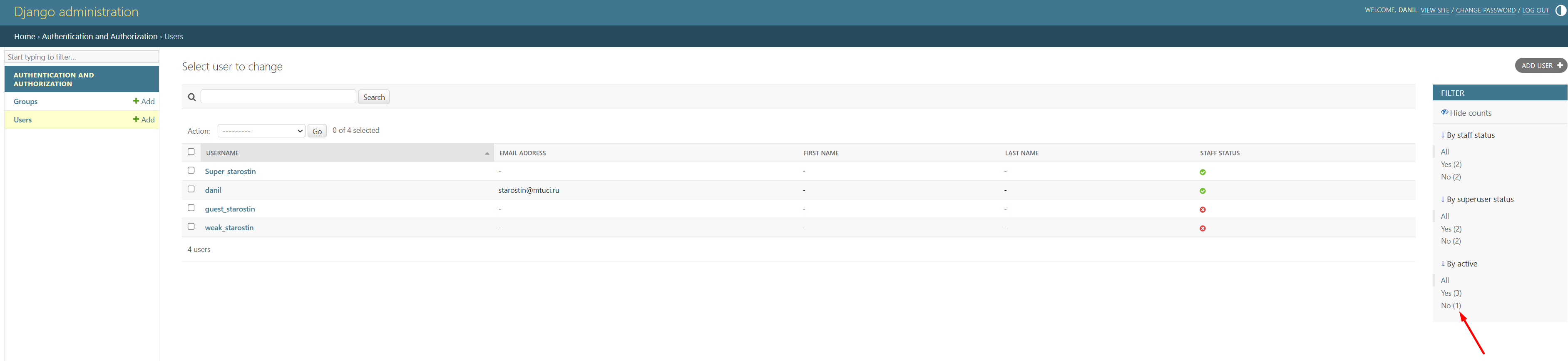












Вывод: рабораторная работа позволила освоить базовые операции с Python и подтвердить корректность установки и настройки Django. Получены навыки работы с встроенными типами данных, функциями и начальная конфигурация веб-приложения.

Лабораторная работа № 2

Создание web-страницы с текстом и работа с шаблонами в Django

**Настройка проекта:**

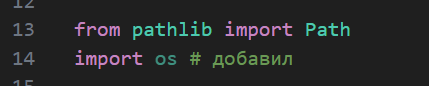
Создаю директорию lab2 и в ней проект firstwebpage командой django-admin.py startproject firstwebpage.

Перехожу в директорию firstwebpage и подключаю приложение flatpages командой python manage.py startapp flatpages

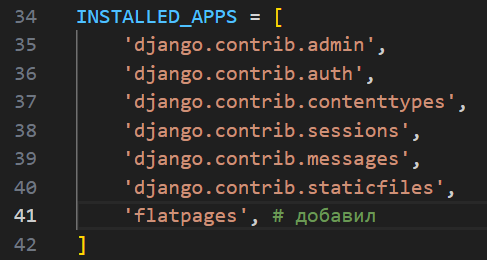


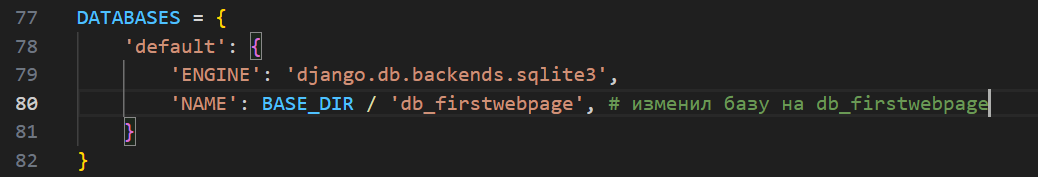
Выполняю базовую настройку (изменение базы данных и migrate).

lab 2\firstwebpage\firstwebpagev\setting.py

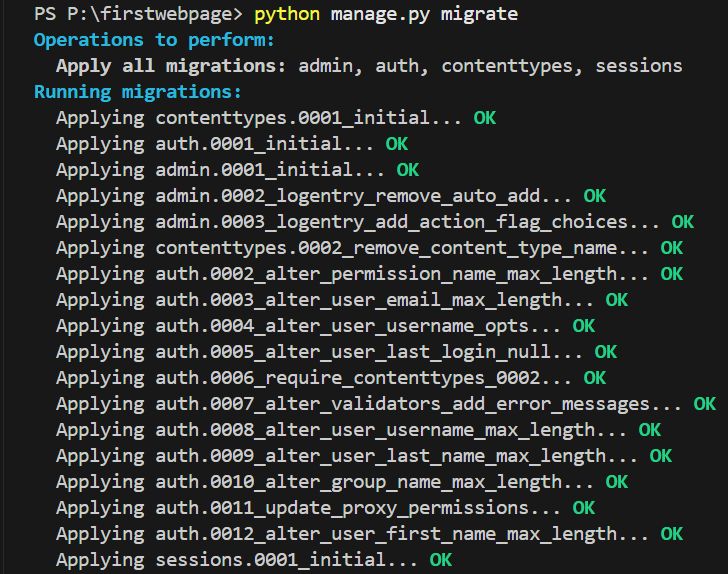


Добавляю 'flatpages' в кортеж INSTALLED\_APPS в файле settings.py.





Выполняю миграцию



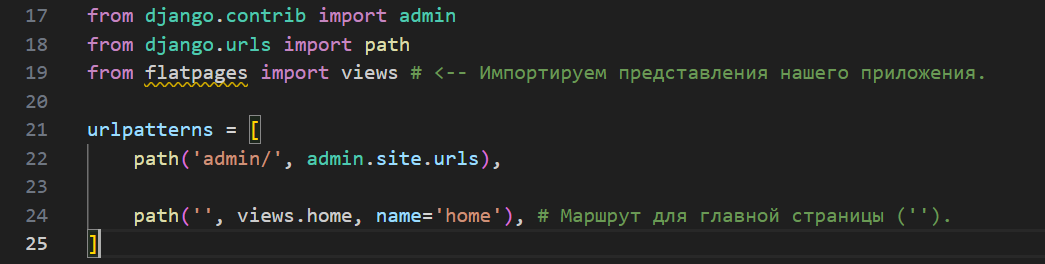
**Создание простого представления:**

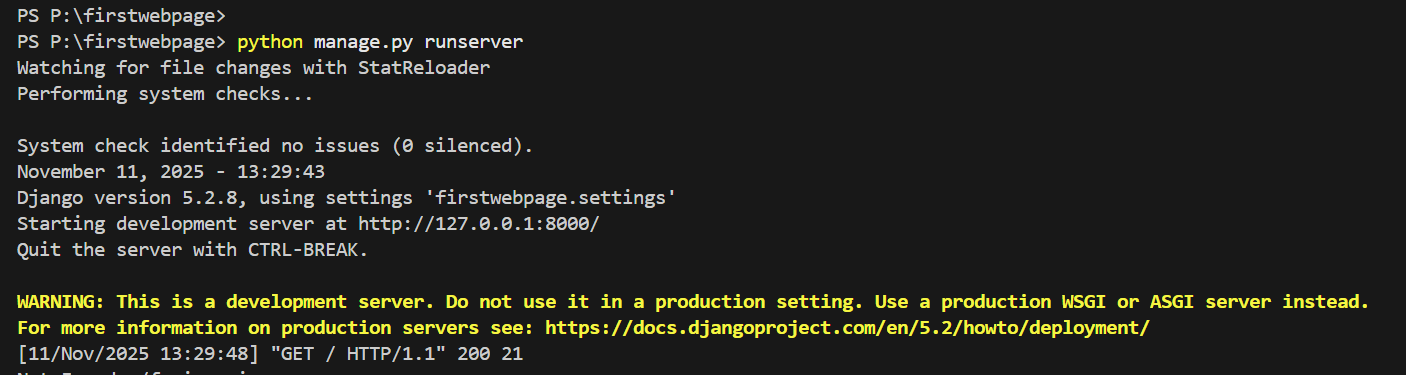
Создаю функцию home в файле views.py приложения flatpages, которая возвращает HttpResponse с текстом 'Привет, Мир!'.

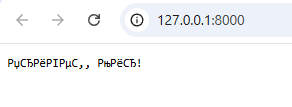


В файле urls.py импортирую views.

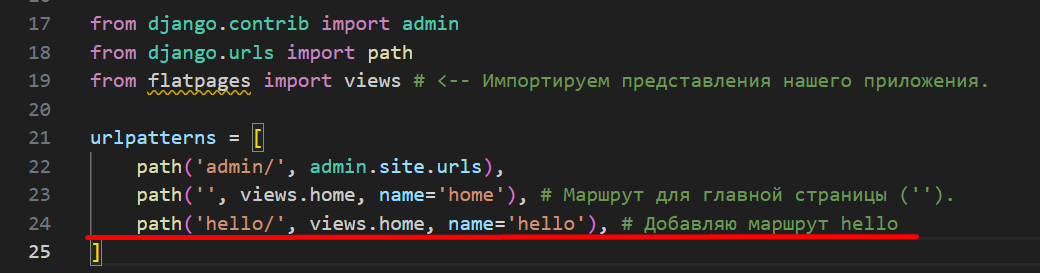
Добавляю новый адрес в urls.py: path('', views.home, name='home').

  
Запускаю сервер и проверяю работу

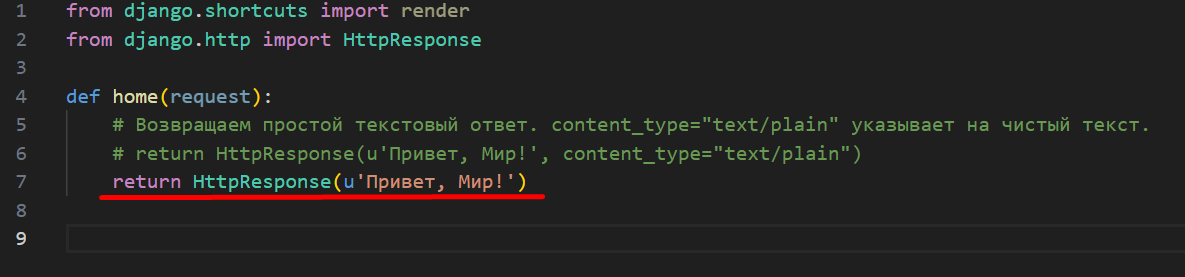




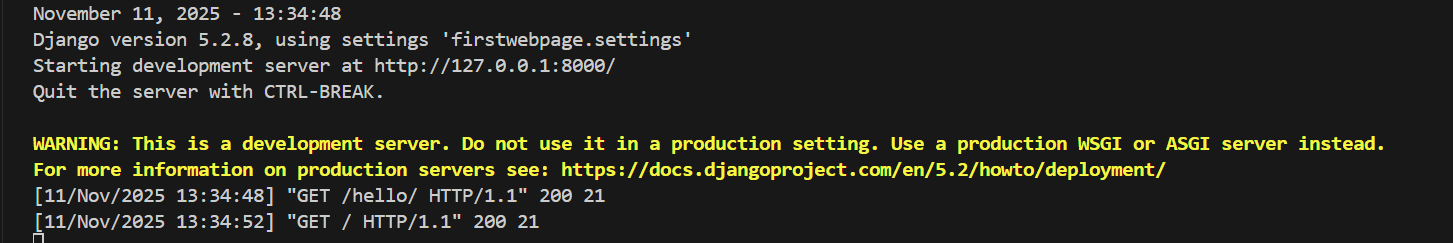
**Задание 1**: Добавляю в urls.py путь, чтобы по адресу http://127.0.0.1:8000/hello/ возвращался тот же текст.

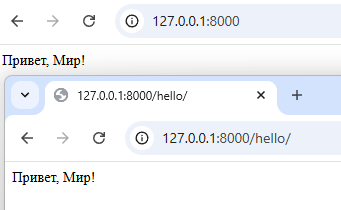


**Задание 2**: Убираю явное указание типа возвращаемого ответа content\_type="text/plain" в HttpResponse.



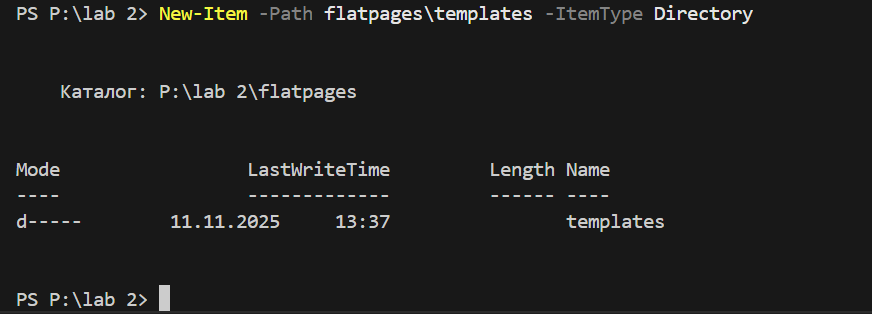
Запускаю сервер и проверяю работу



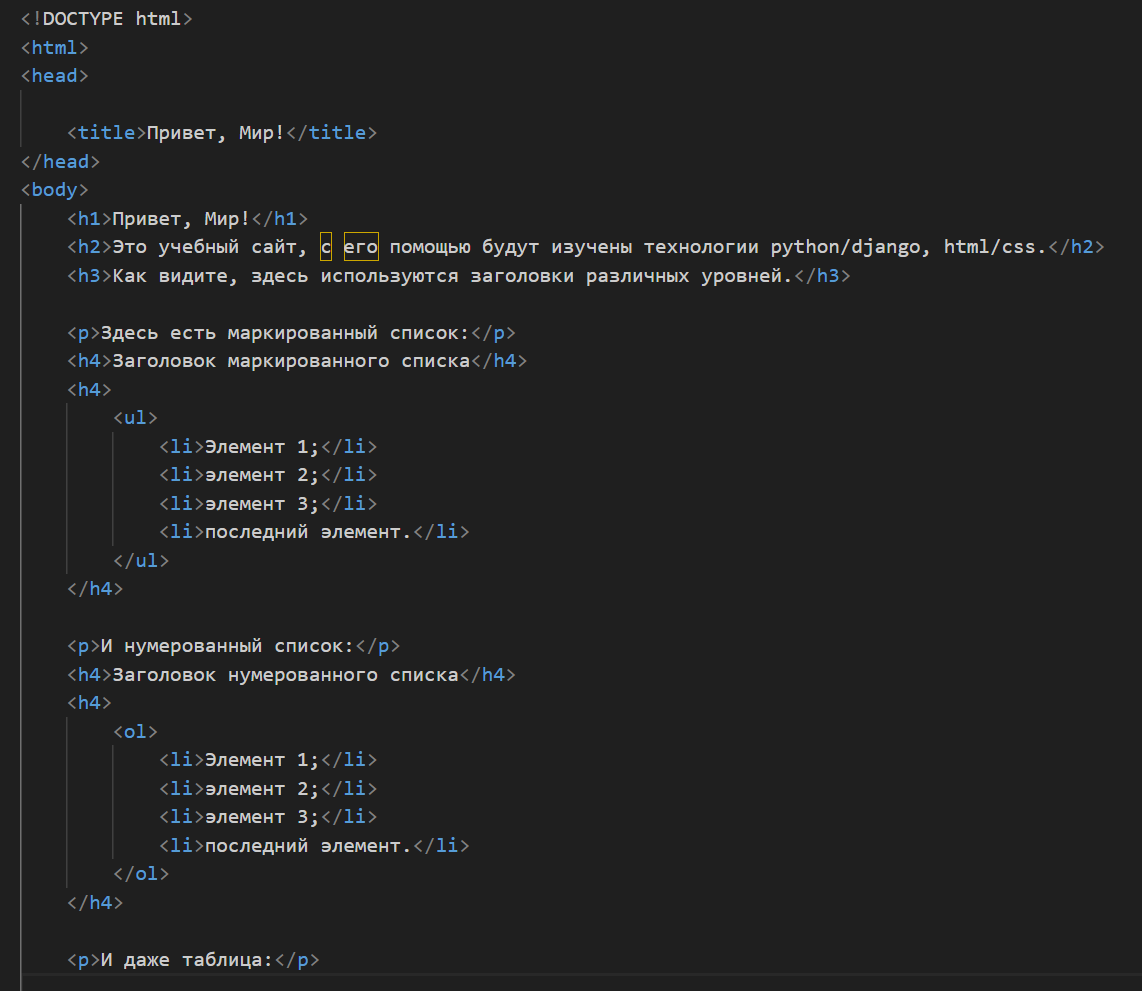


**Работа с шаблонами**:

Создаю папку templates в директории flatpages.

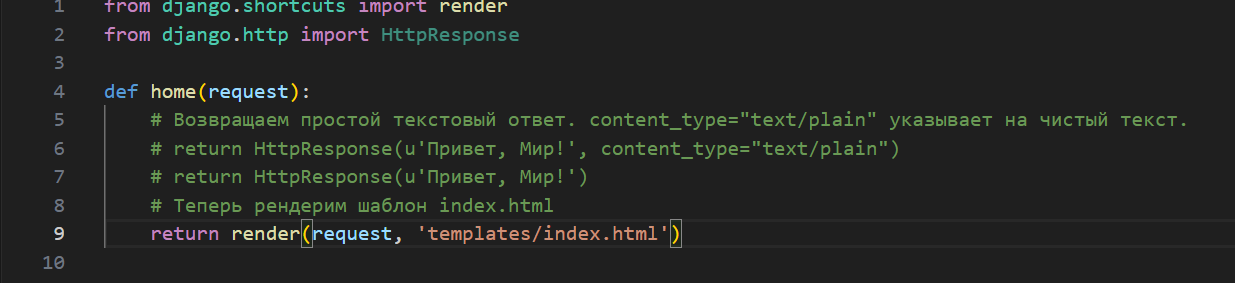


Создаю файл index.html в папке templates с требуемой HTML-разметкой (заголовки, маркированный и нумерованный списки, таблица) .

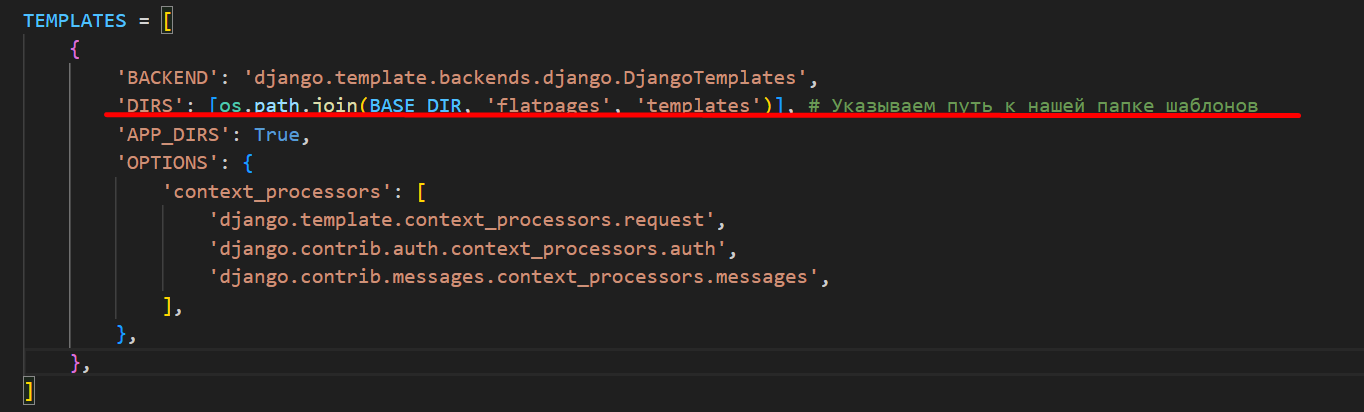


В файле views.py импортирую render.

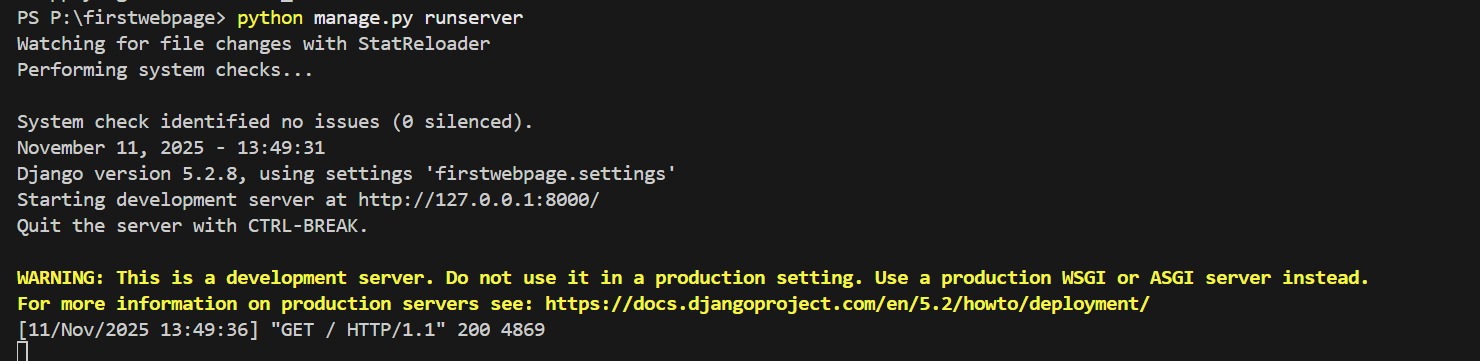
Изменяю функцию home для использования шаблона: return render(request, 'templates/index.html').

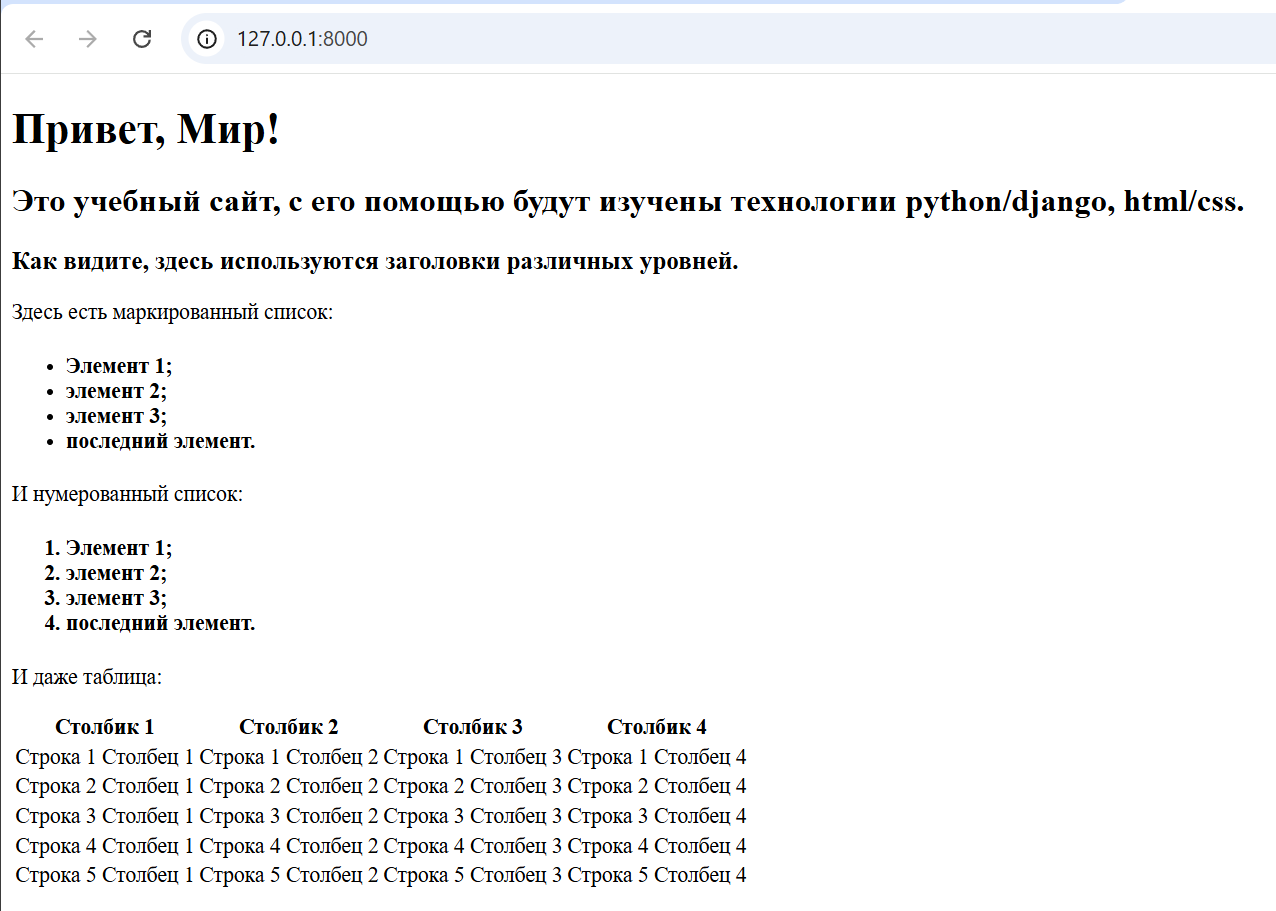


В файле settings.py изменяю поле DIRS кортежа TEMPLATES на 'DIRS': [os.path.join(BASE\_DIR, 'flatpages'),], для поиска шаблона.



Проверяю отображение страницы по адресу http://127.0.0.1:8000/.





**Задание по шаблону**:

Копирую index.html и создаю абсолютно идентичный шаблон static\_handler.html.

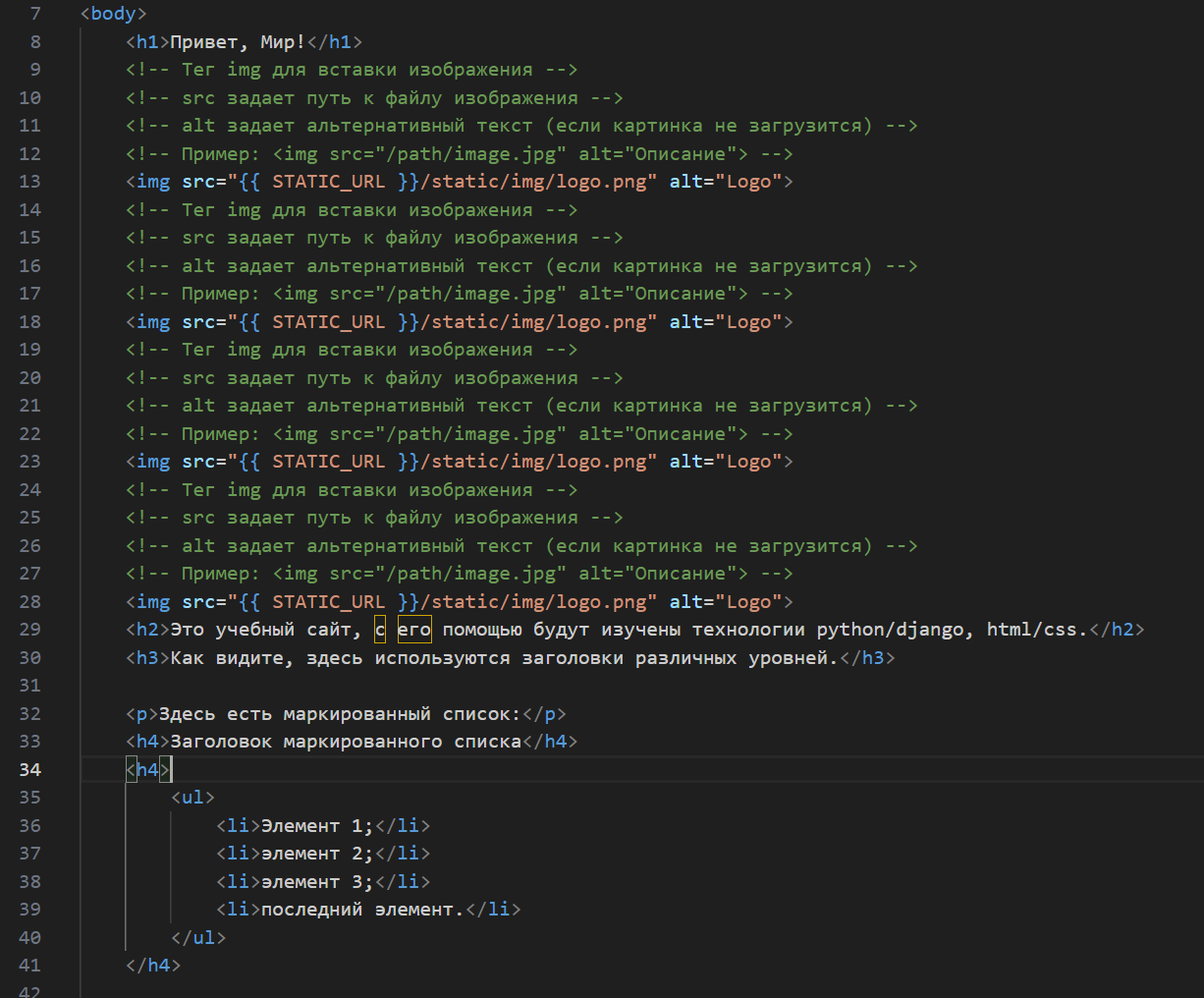
В static\_handler.html:

Добавляю 2 строки и 2 столбца к существующей таблице.

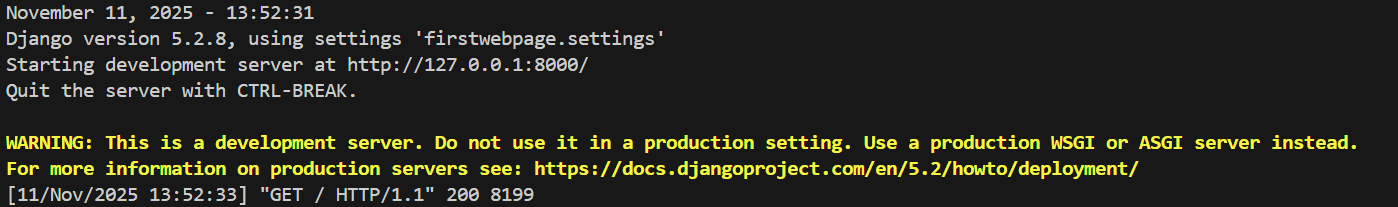
Добавляю границы для таблицы (используя CSS стили в теге <table> или далее через CSS-файл).

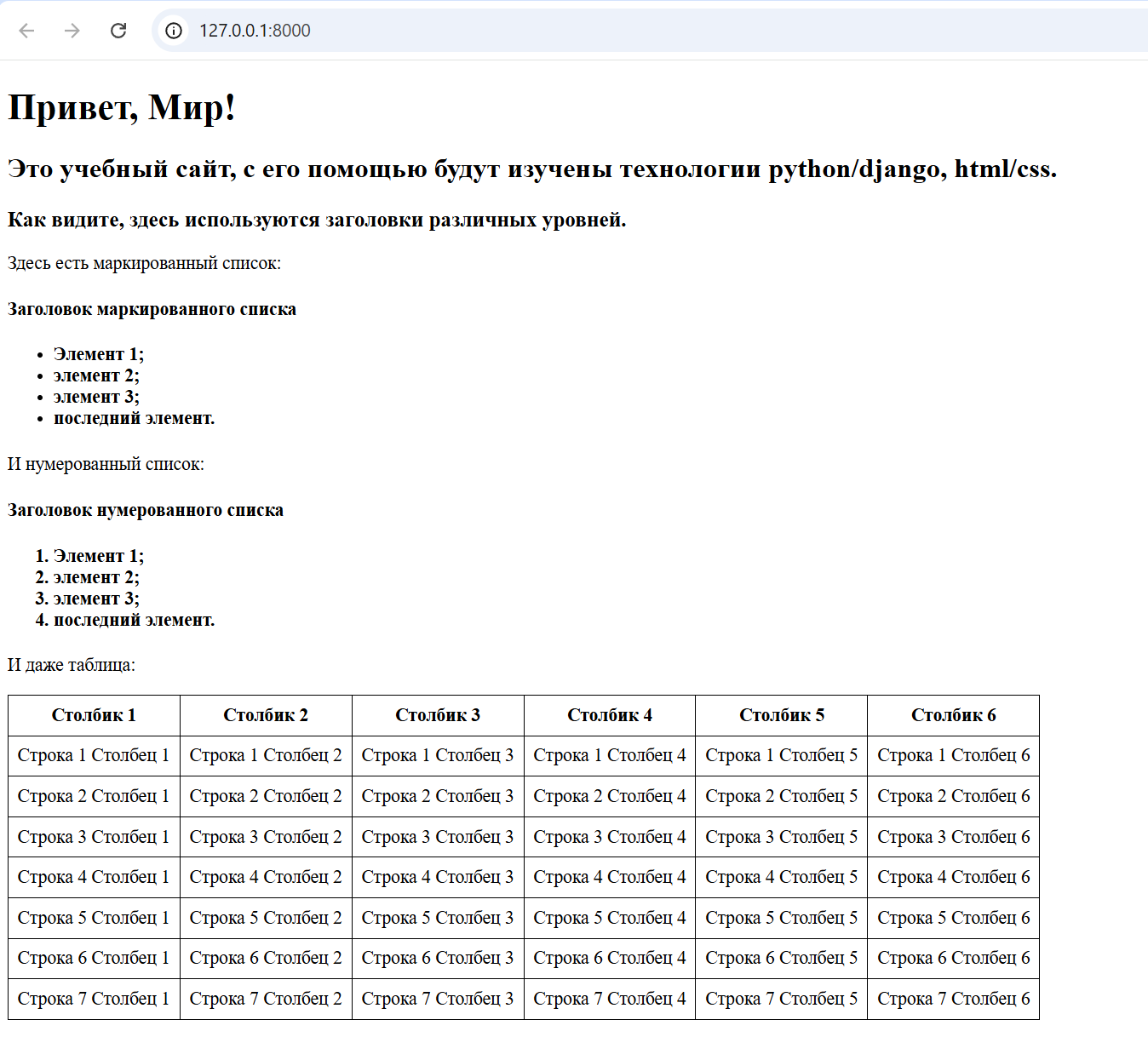


Изменяю заголовки списков (нумерованного и маркированного) на подзаголовки четвертого уровня <h4>.



Запускаю сервер





**Настройка статичных файлов (CSS):**

Создаю папку static в директории flatpages.

Создаю в ней файл index.css с указанными стилями (фон, шрифты, таблица и т.д.) .



В файл static\_handler.html, внутри тега <head>, вставляю тег для подключения CSS: <link rel="stylesheet" href="{{ STATIC\_URL }}/static/index.css">.

**Задание по стилям**:

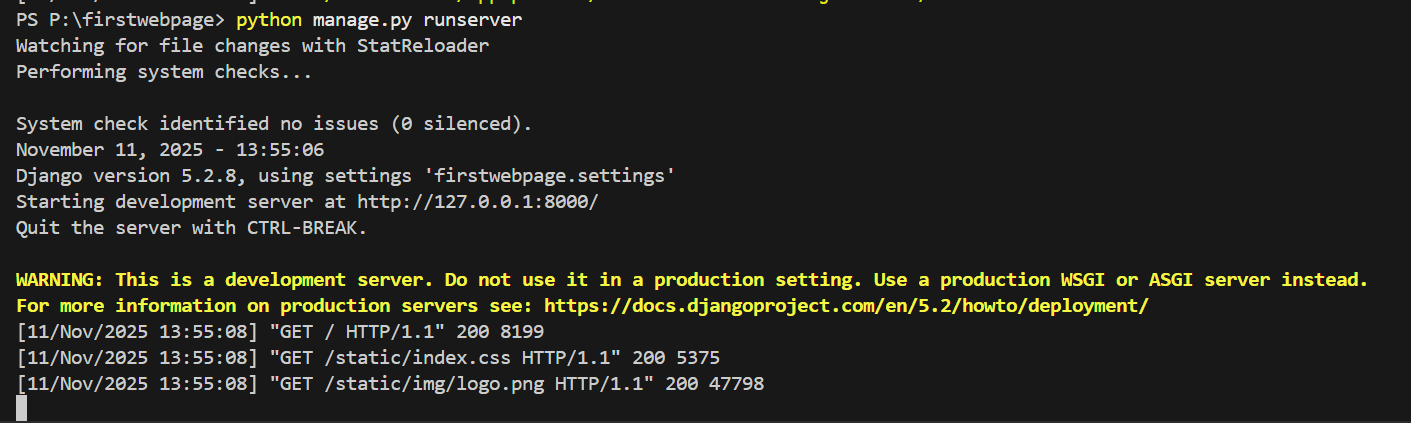
В index.css:

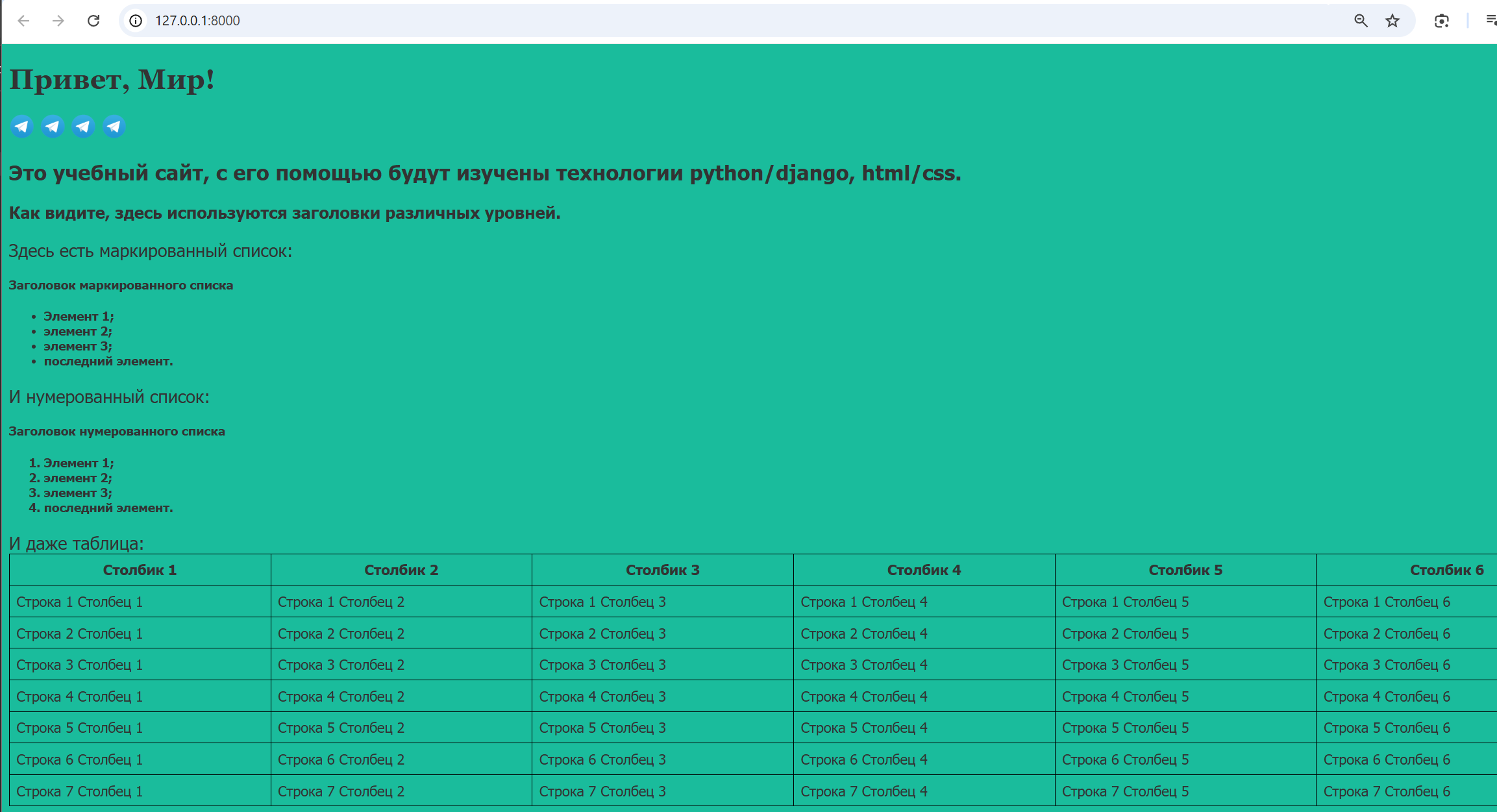
Устанавливаю для заголовка первого уровня (h1) шрифт с засечками.

Добавляю в static\_handler.html картинку и делаю ее высоту 30px (согласно стилю img в index.css).

Изменяю размер шрифта для подзаголовков четвертого уровня (h4).

Устанавливаю ширину таблицы на 100% экрана, используя стиль table { width: 100%; } в index.css.





Загружаю проект на гит-репозиторий.

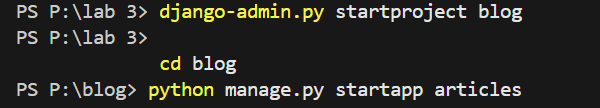
Лабораторная работа № 3

Создание первой модели данных и ее регистрация в административном приложении Django

**Настройка проекта и приложения:**

Создаю директорию lab3 и в ней проект blog командой django-admin.py startproject blog.

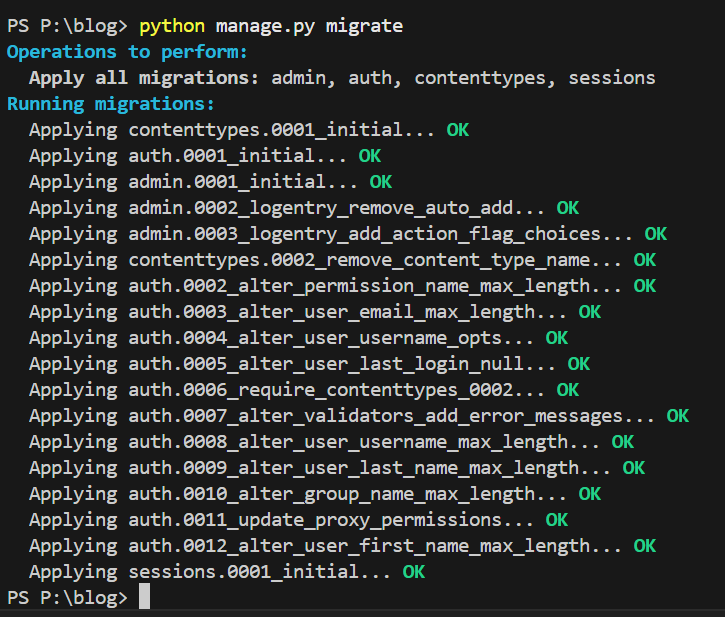
Перехожу в папку blog и создаю приложение articles командой python manage.py startapp articles.

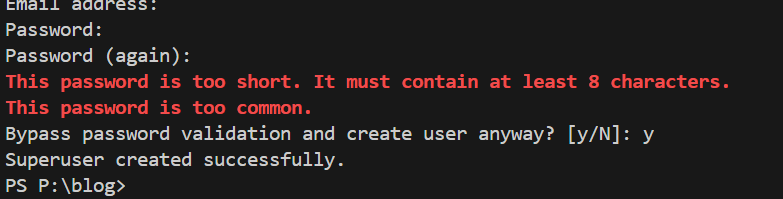


Выполняю базовую настройку проекта (добавление articles в INSTALLED\_APPS,



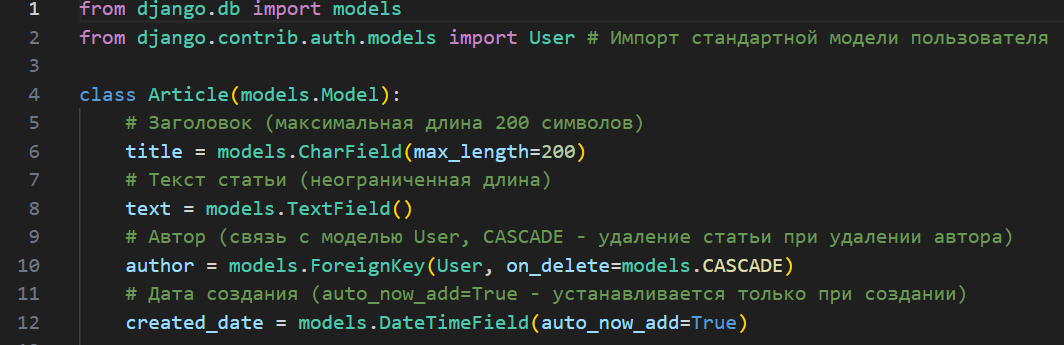
настройка базы данных, migrate,

 создание суперпользователя.



**Создание и регистрация модели**:

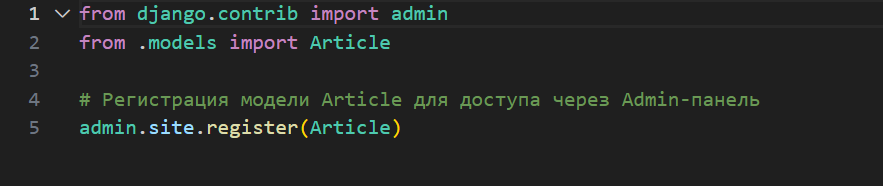
В файле models.py директории articles создаю модель Article с полями title, author, text, и created\_date, включая методы \_\_unicode\_\_ и get\_excerpt .



В файле admin.py регистрирую модель Article в административном интерфейсе:

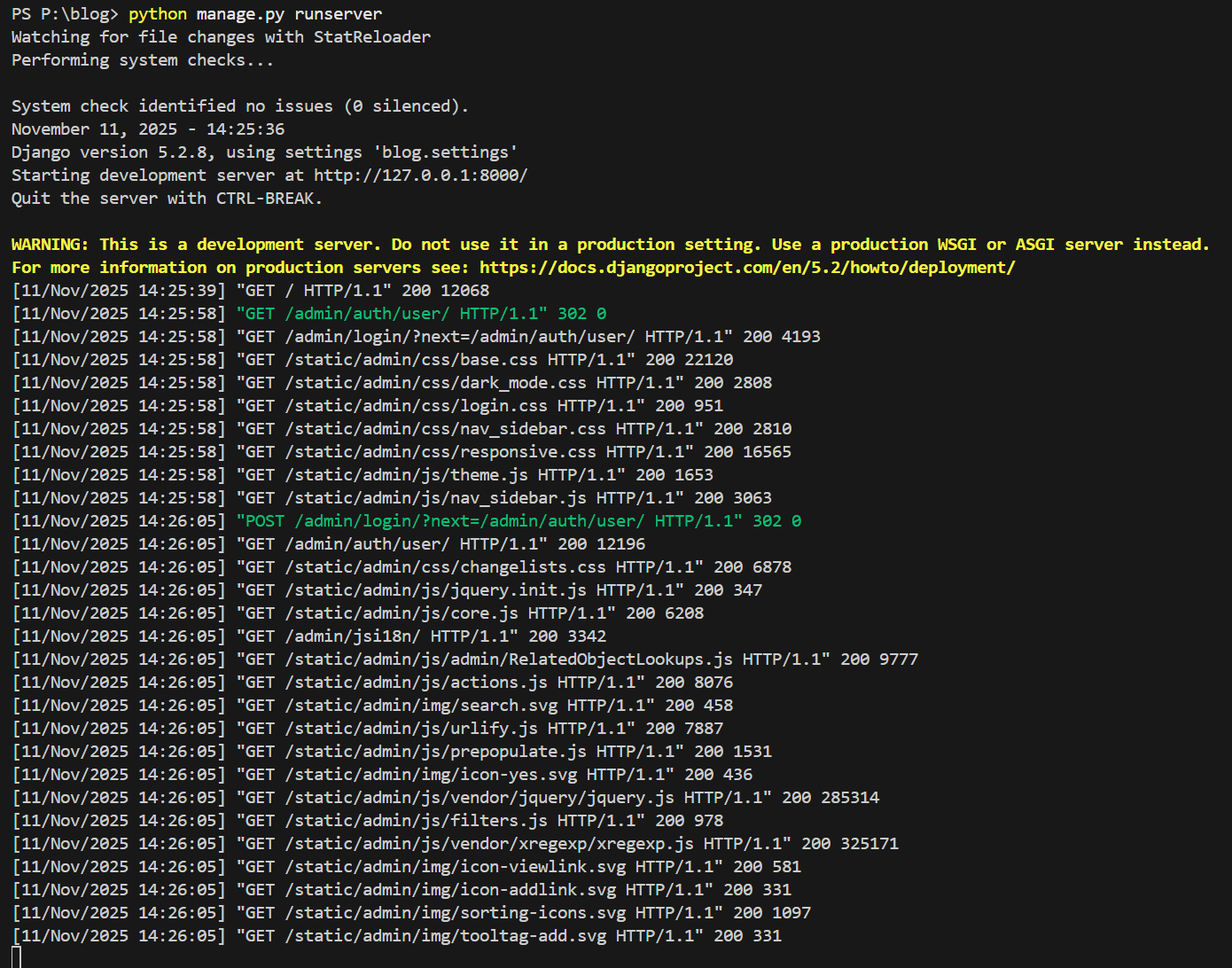
Создаю класс ArticleAdmin для описания отображения.

Вызываю admin.site.register(Article, ArticleAdmin).

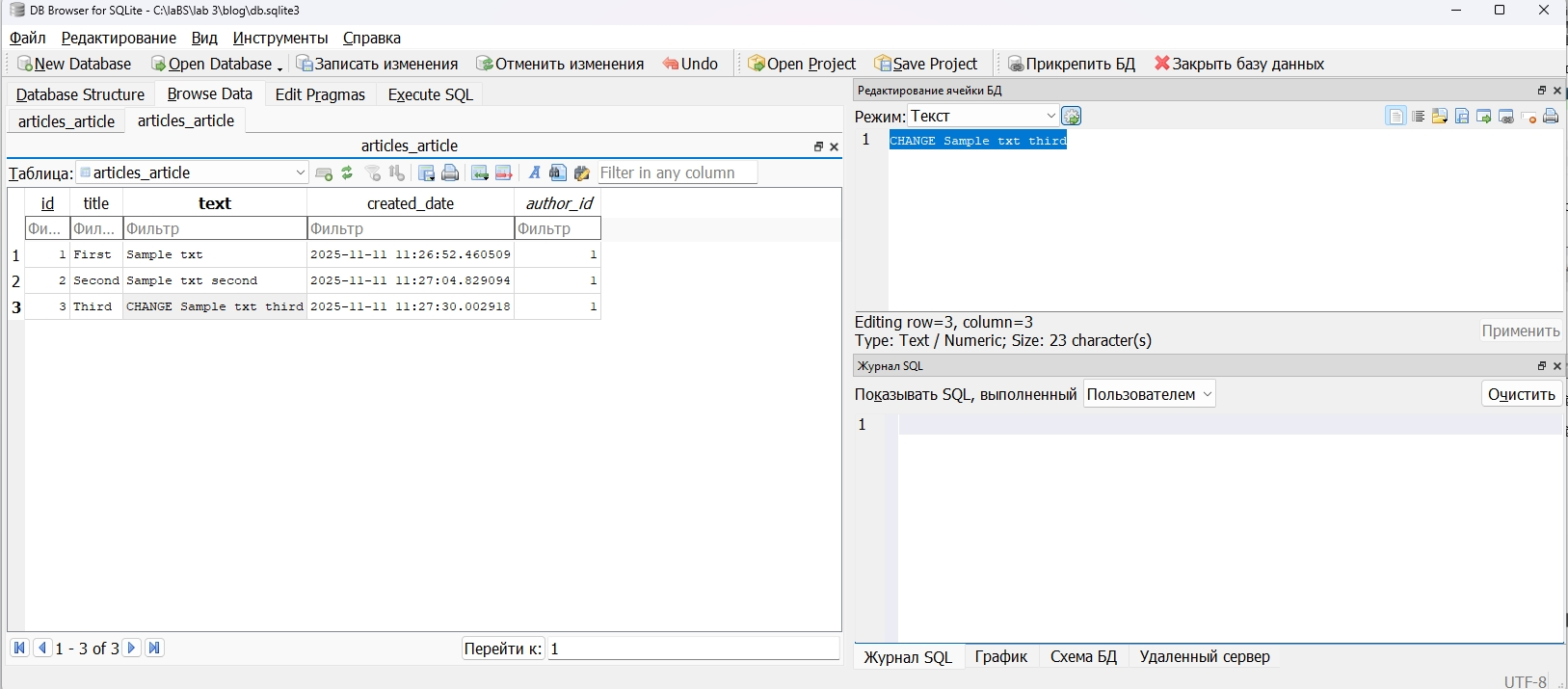


**Задание по административной панели**:

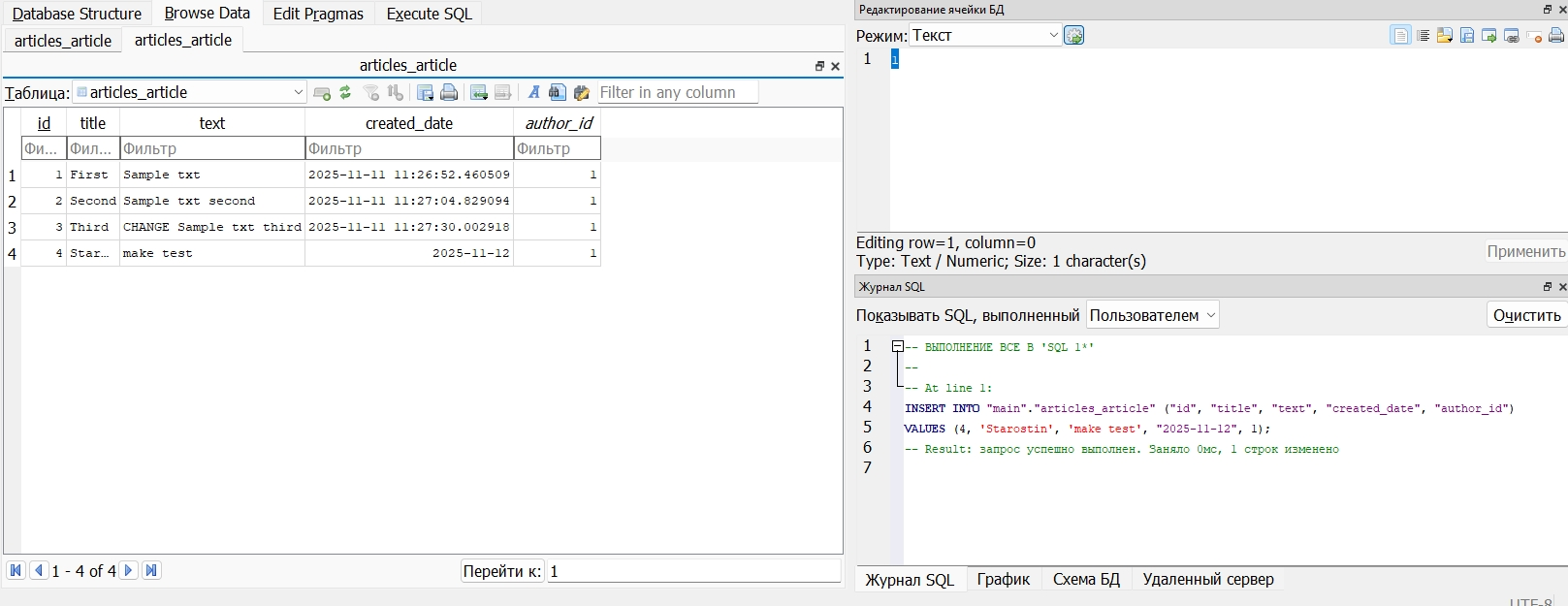
Запускаю сервер и перехожу в административную панель <http://127.0.0.1:8000/admin/>.



Перехожу во вкладку Articles и создаю 3 статьи, заполняя все поля.



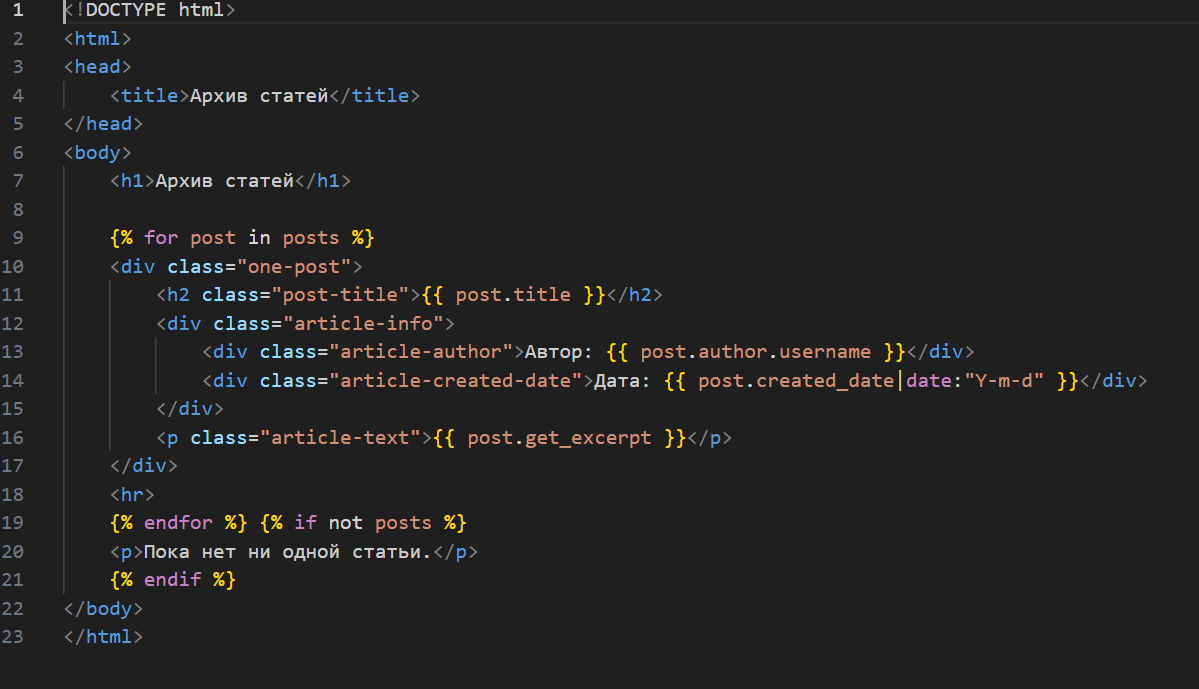
С помощью интерфейса управления статьями, нахожу созданные записи, изменяю текст записи.



**Динамическое генерирование шаблона (Архив статей):**

В директории articles создаю папку templates, а в ней файл archive.html.

В archive.html устанавливаю <title> как "Архив статей".

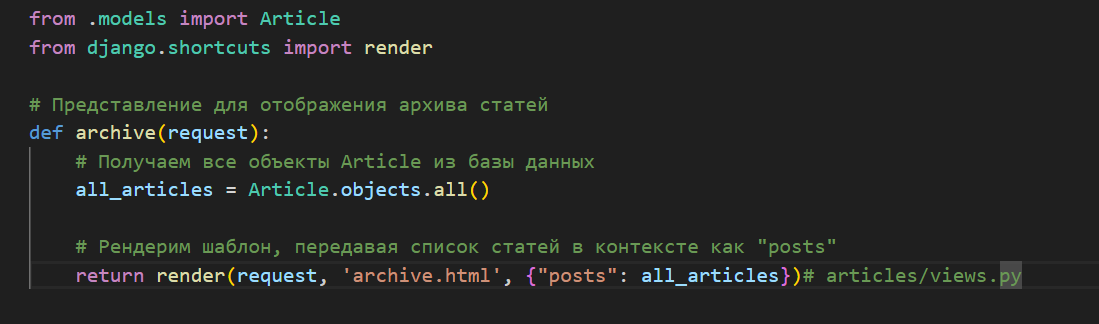


Создаю в <body> теги <div class="header"> и <div class="archive"> .

В <div class="header"> добавляю изображение: <img src="{{ STATIC\_URL }}img/logo.png" />.

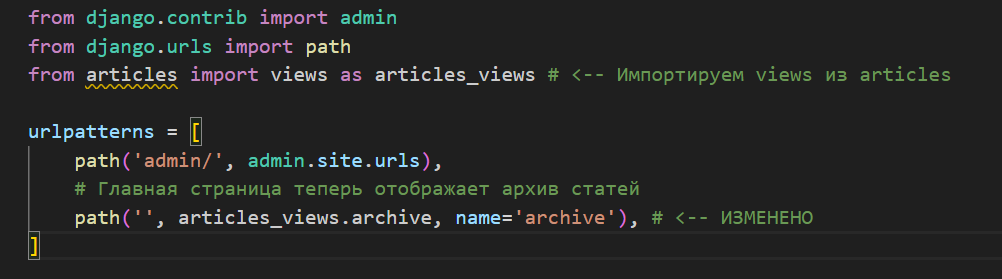
В <div class="archive"> добавляю шаблон с циклом для отображения статей:

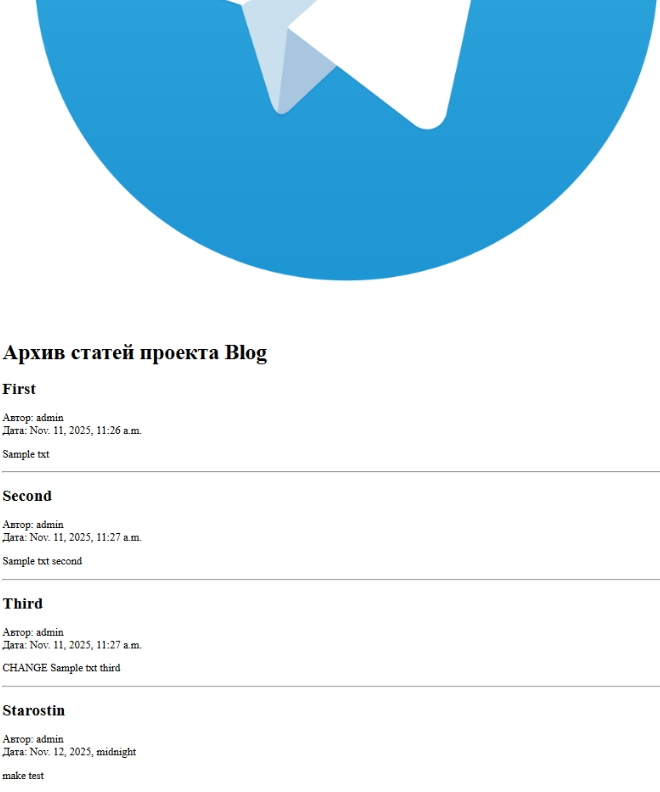
В файле views.py приложения articles создаю представление archive.



Реализую функцию archive для получения всех статей и их рендеринга с шаблоном: return render(request, 'archive.html', {"posts": Article.objects.all()}).

Задание по URL:

Самостоятельно настраиваю urls.py так, чтобы при заходе на главную страницу (/) отображался список всех записей, используя представление archive. 



Загружаю проект на гит-репозиторий.