**Лабораторные работы**

**(Python 3.7 + Django)**

Оглавление

[**Лабораторная работа № 1** 3](#_Toc5009964)

[**Лабораторная работа №2** 9](#_Toc5009965)

[**Лабораторная работа № 3** 16](#_Toc5009966)

[**Лабораторная работа № 4** 21](#_Toc5009967)

[**Лабораторная работа № 5** 26](#_Toc5009968)

[**Лабораторная работа №6** 29](#_Toc5009969)

[**Лабораторная работа №7** 32](#_Toc5009970)

# **Лабораторная работа № 1**

*Начало работы с Python*

Python – это интерпретируемый язык программирования высокого уровня, созданный Гвидо ван Россумом и впервые выпущенный в 1991 году.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных.

Python может служить скриптовым языком для создания веб-приложений. Веб-фреймворки, такие как Django, Pylons, Pyramid, TurboGears, web2py, Tornado, Flask, Bottle и Zope, помогают разработчикам в разработке и сопровождении сложных приложений.

Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года.

Установка Python 3.7.X (неважно, какая цифра в версии будет стоять третьей):

Необходимо скачать с официального сайта https://www.python.org/downloads/ последнюю версию интерпретатора Python для нужной операционной системы, затем запустить установочный файл.

Узнать точную установленную версию Python3 можно введя данную команду в командной строке: python3 -V

Создайте директорию для лабораторных работ (в названии директории не используйте кириллицу, т.к. при выполнении команд может произойти ошибка). В созданной папке создайте файл «helloworld.py». Используя редактор кода IDLE или любой другой добавьте в созданный файл следующий код: print("Hello world!")

Для запуска программы необходимо открыть командную строку (cmd – в Windows, Terminal – в macOS), перейти в консоли в папку с лабораторными работами с помощью команды cd + адрес директории. Например: cd Users/admin/Desktop/labs\_python. После перехода в нужную директорию, нужно запустить скрипт с помощью команды: python3 helloworld.py

*Пример:* 

Рисунок 1. Пример работающего скрипта

Таким образом, первая программа на Python3 запущена.

Затем в той же директории создайте файл «mygroup.py», в котором будут находится данные о студентах вашей группы (достаточно 5-7 человек) в формате списка. Обратите внимание, что весь список сохраняется в одну переменную (например, groupmates), для каждого студента должны быть заполнены поля: имя, фамилия, список экзаменов, оценки.

*Пример:*

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]

}

]

**Задание.** Вам необходимо написать функцию фильтрации студентов по средней оценке, так, чтобы на экран выводился список студентов, средний балл которых выше заданного. Средний балл, по которому будет проводиться фильтрация, вводится пользователем с клавиатуры.

*Пример функции вывода списка всех студентов:*

def print\_students(students):

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

for student in students:

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

print\_students(groupmates)

Функция в Python – это объект, принимающий аргументы и возвращающий значение. Разберем функцию вывода подробнее.

def print\_students(students): – объявление функции.

def – стандартная инструкция для объявлении функции.

print\_students – название функции (может быть любым).

(students) – принимаемое значение.

print (u"Имя".ljust(15), u"Фамилия".ljust(10), u"Экзамены".ljust(30), u"Оценки".ljust(20)) – вывод на экран «шапки» таблицы.

u"Имя" – префикс u обозначает, что данная строка будет выведена в кодировке Unicode.

.ljust(15) – метод строки, который добавляет нужное количество пробелов, чтобы длина строки стала равной передаваемому параметру. Используется для выравнивания строк.

for student in students:

print(student["name"].ljust(15), student["surname"].ljust(10), str(student["exams"]).ljust(30), str(student["marks"]).ljust(20)) – цикл, в котором построчно выводится информация о каждом студенте.

print\_students(groupmates) – вызов функции. В качестве передаваемого значения – список всех студентов (groupmates).

*Начало работы с Django*

Django – это свободный веб-фреймворк с открытым исходным кодом на основе Python, который следует архитектурному шаблону MVC (Model-View-Controller: Модель-Представление-Контроллер).

Установка Django:

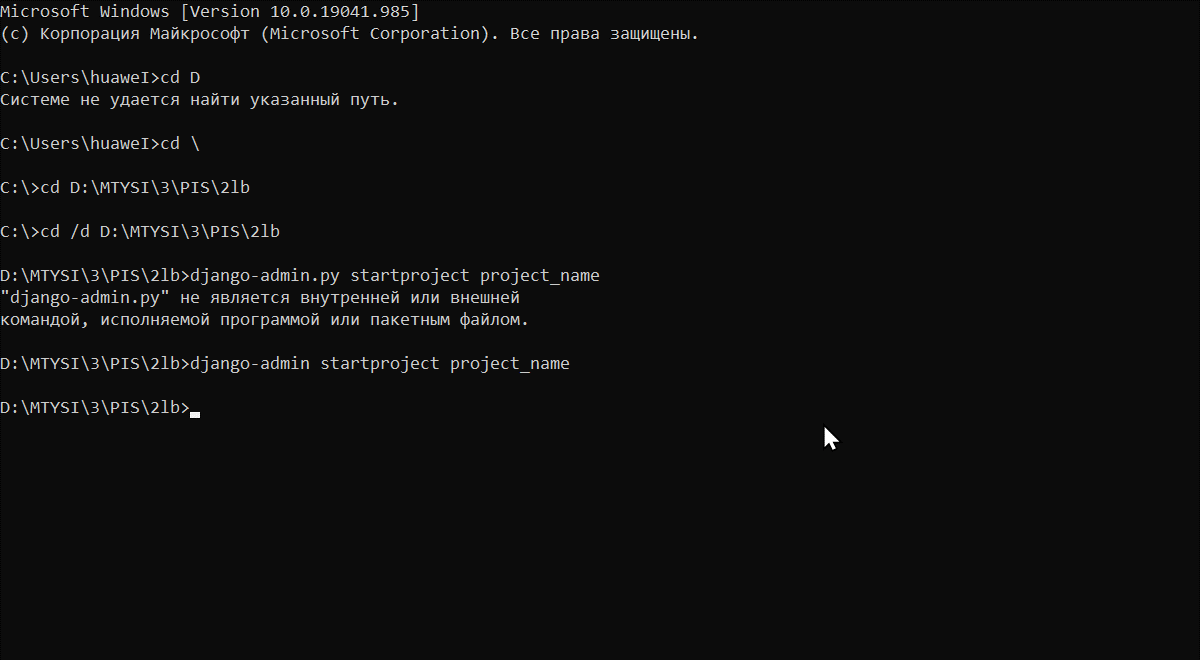
Необходимо перейти на официальный сайт фреймворка Django https://www.djangoproject.com/download/ , где будет написана команда для установки с указанием последней версии, например, на момент написания учебного пособия актуальной версией Django была версия 2.1.7. Пример установки фреймворка Django с помощью pip (системы управления пакетами): pip3 install Django==2.1.7

***Примечание:*** *команда pip3 используется только для операционных систем macOS и Linux, для Windows используется команда pip.*

После установки фреймворка приступим к созданию проекта. Для этого с помощью командной строки необходимо перейти в директорию, в которой находятся ваши лабораторные работы, и выполнить команду:

django-admin.py startproject project\_name

project\_name – имя вашего проекта

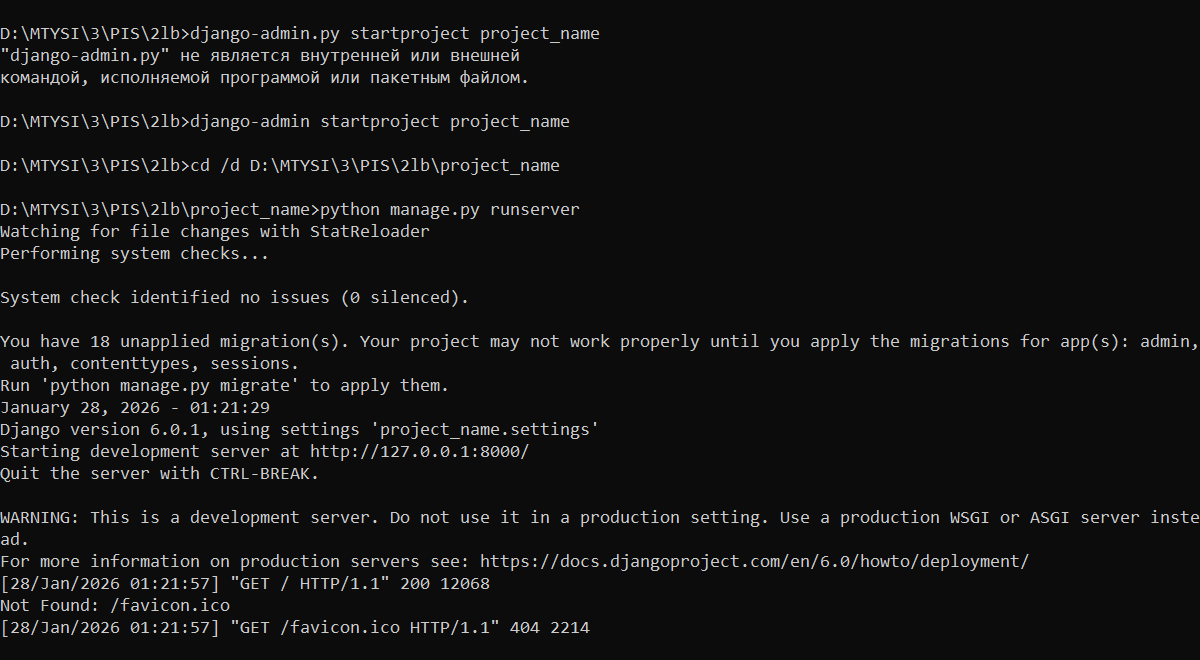


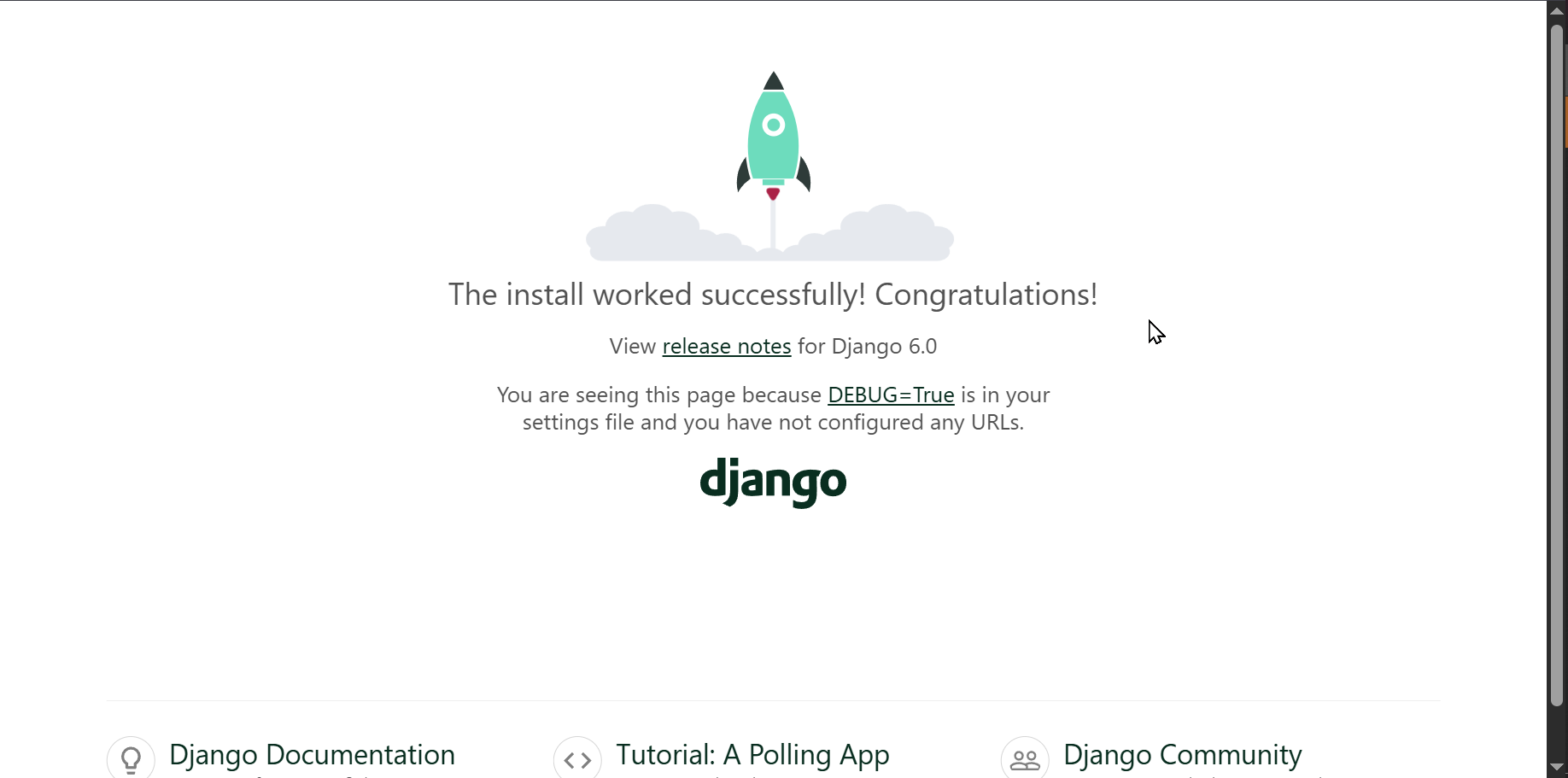
После выполнения данной команды в выбранной директории создается папка project\_name, в которой содержится:

* Файл manage.py – файл для командной строки (терминала), который позволяет взаимодействовать с проектом с помощью различных методов.
* Файл \_\_init\_\_.py – файл необходим для того, чтобы Python рассматривал данный каталог как пакет, т.е., как группу модулей. Это пустой файл.
* Файл settings.py – в нем содержатся настройки для текущего проекта.
* Файл urls.py – описания URL для текущего проекта. «Оглавление» сайта.
* Файл wsgi.py – файл для настроек текущего проекта. Не изменяется.

Для того, чтобы проверить, правильно ли был установлен фреймворк Django, необходимо запустить локальный web-сервер разработки. Для запуска сервера перейдите в директорию project\_name, в которой находится файл manage.py и выполните команду, которая запускает локальный сервер на порту 8000:

python manage.py runserver или python3 manage.py runserver





Если при установке не возникло ошибок, то в консоли появится следующее:



Рисунок 2. Сообщение о правильном запуске сервера

Затем перейдите по адресу: <http://127.0.0.1:8000/> Если сервер удачно запущен, то будет выведено сообщение: The install worked successfully! Congratulations!

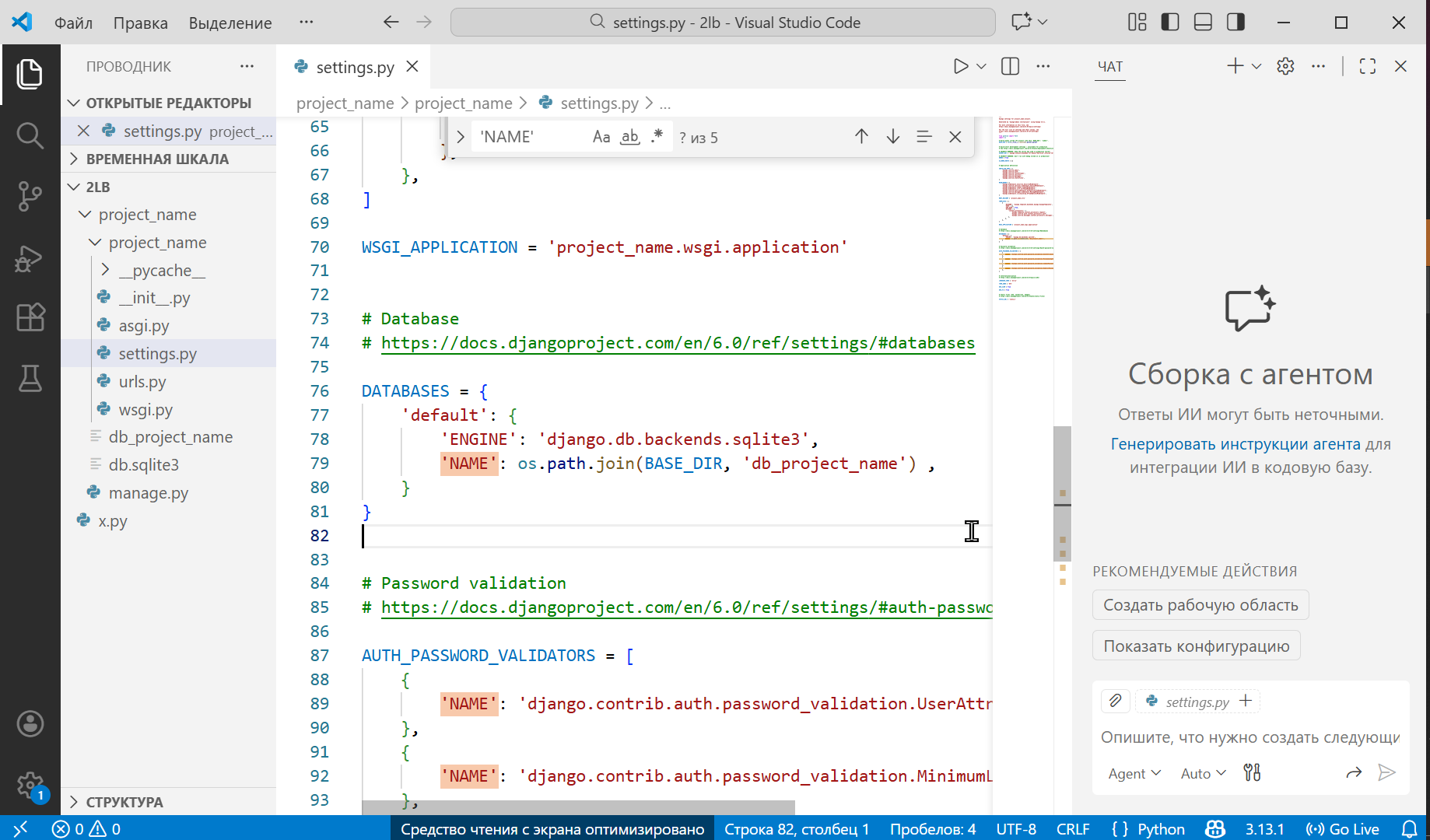
Для того, чтобы попасть в административный интерфейс Django, необходимо обладать правами суперпользователя. Для этого в свою очередь необходимо создать нужные таблицы в базе данных, в которой будут храниться все данные о пользователях.

**Задание.**

В файле settings.py измените параметр базы данных. Переменной DATABASES есть внутренний словарь с ключами ENGINE и NAME. Для параметра NAME установите следующее значение:

'NAME': os.path.join(BASE\_DIR, 'db\_project\_name')

'db\_project\_name' – данный параметр является названием базы данных, так что он может быть любым.

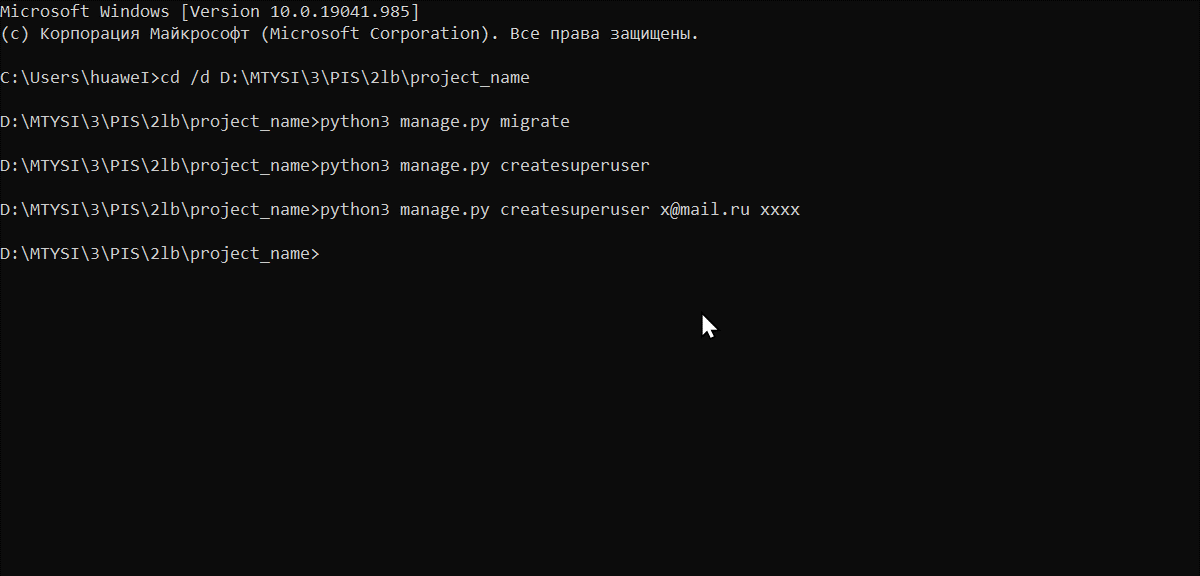


Чтобы создать таблицы базы данных, необходимо перейти в директорию project\_name и через командную строку выполнить команду:

python3 manage.py migrate

Затем создаем суперпользователя (при создании необходимо ввести логин, email и пароль пользователя):

python3 manage.py createsuperuser

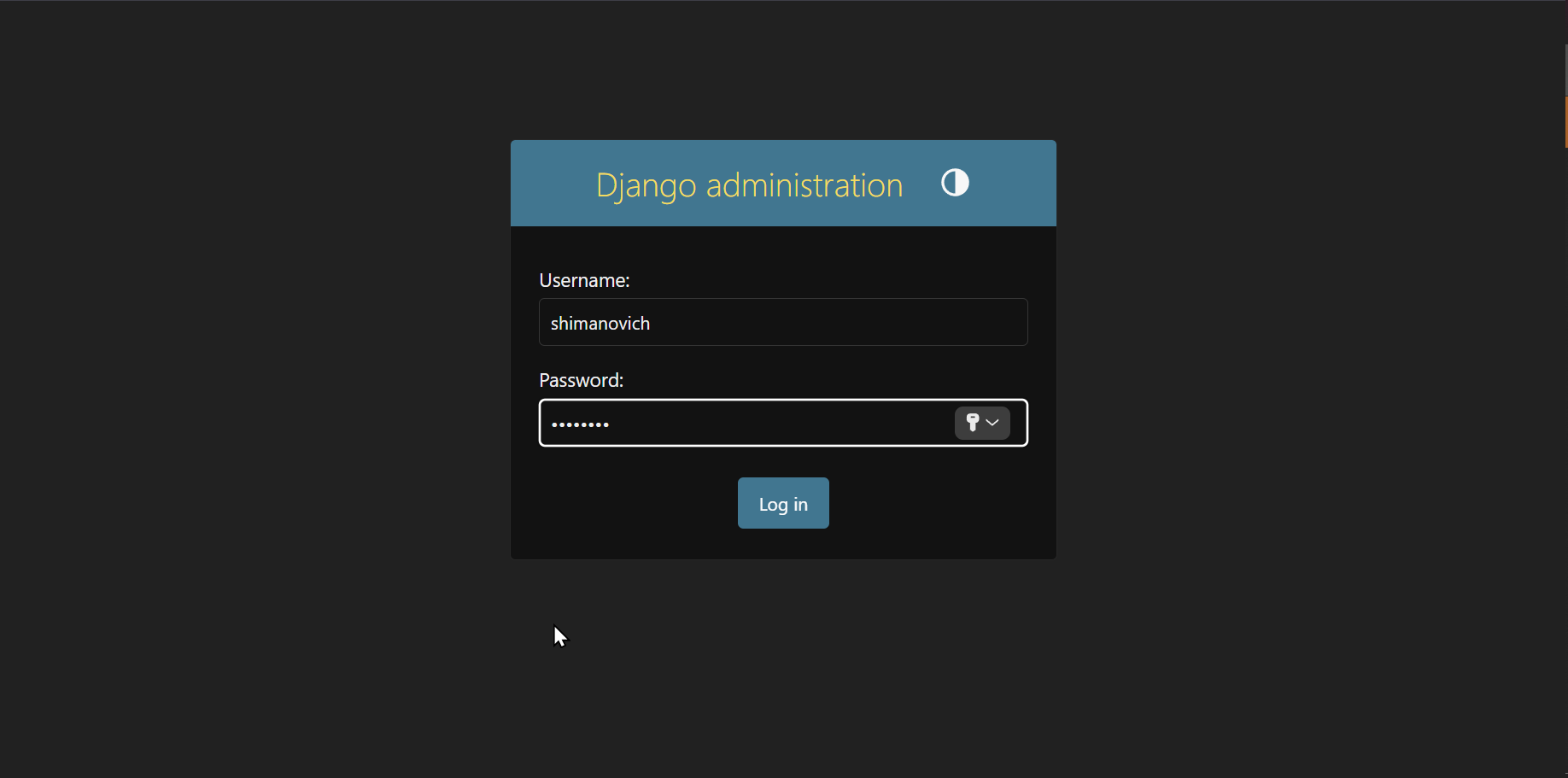


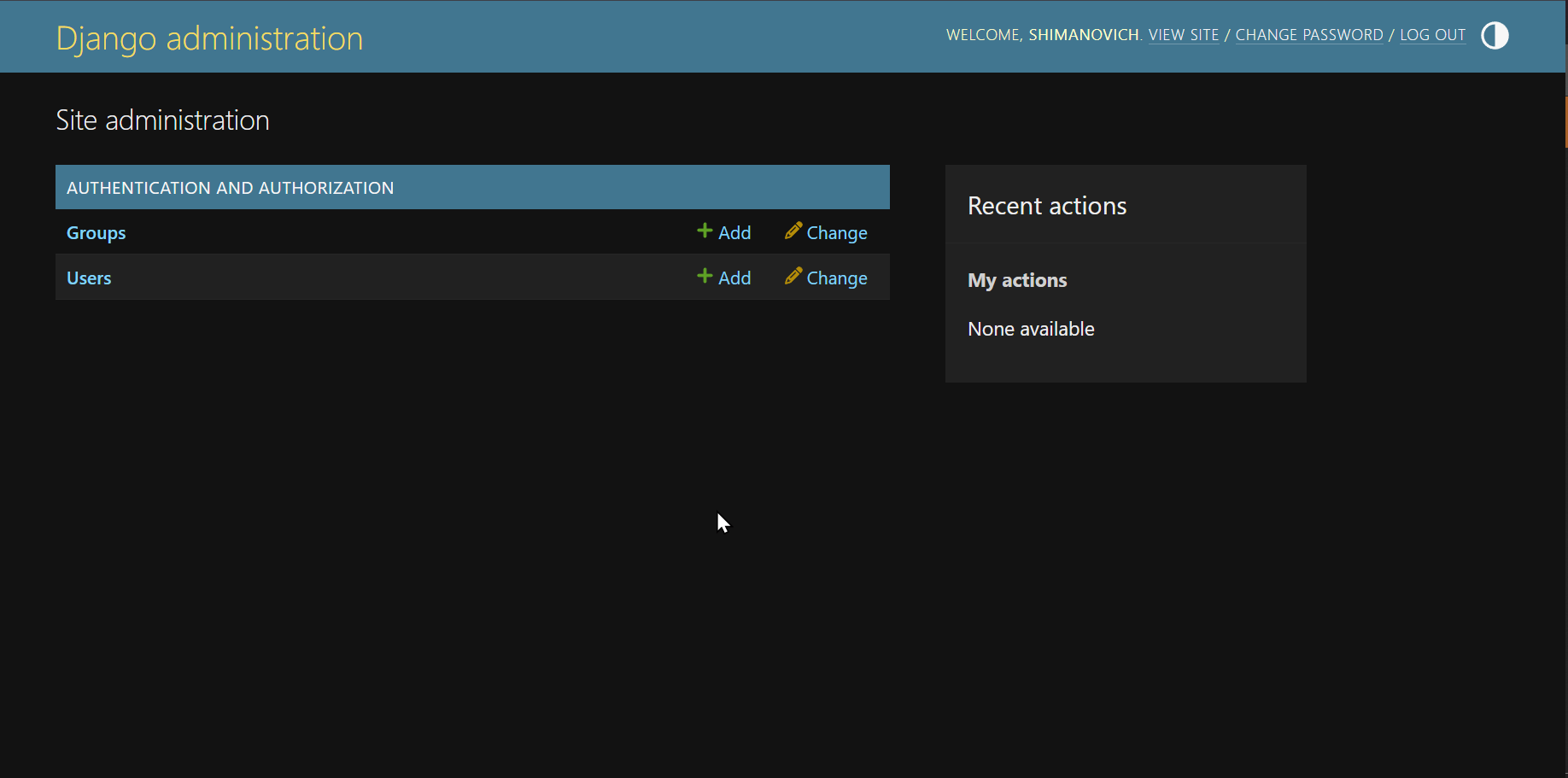
Теперь необходимо запустить сервер, если он не был запущен до этого, командой:



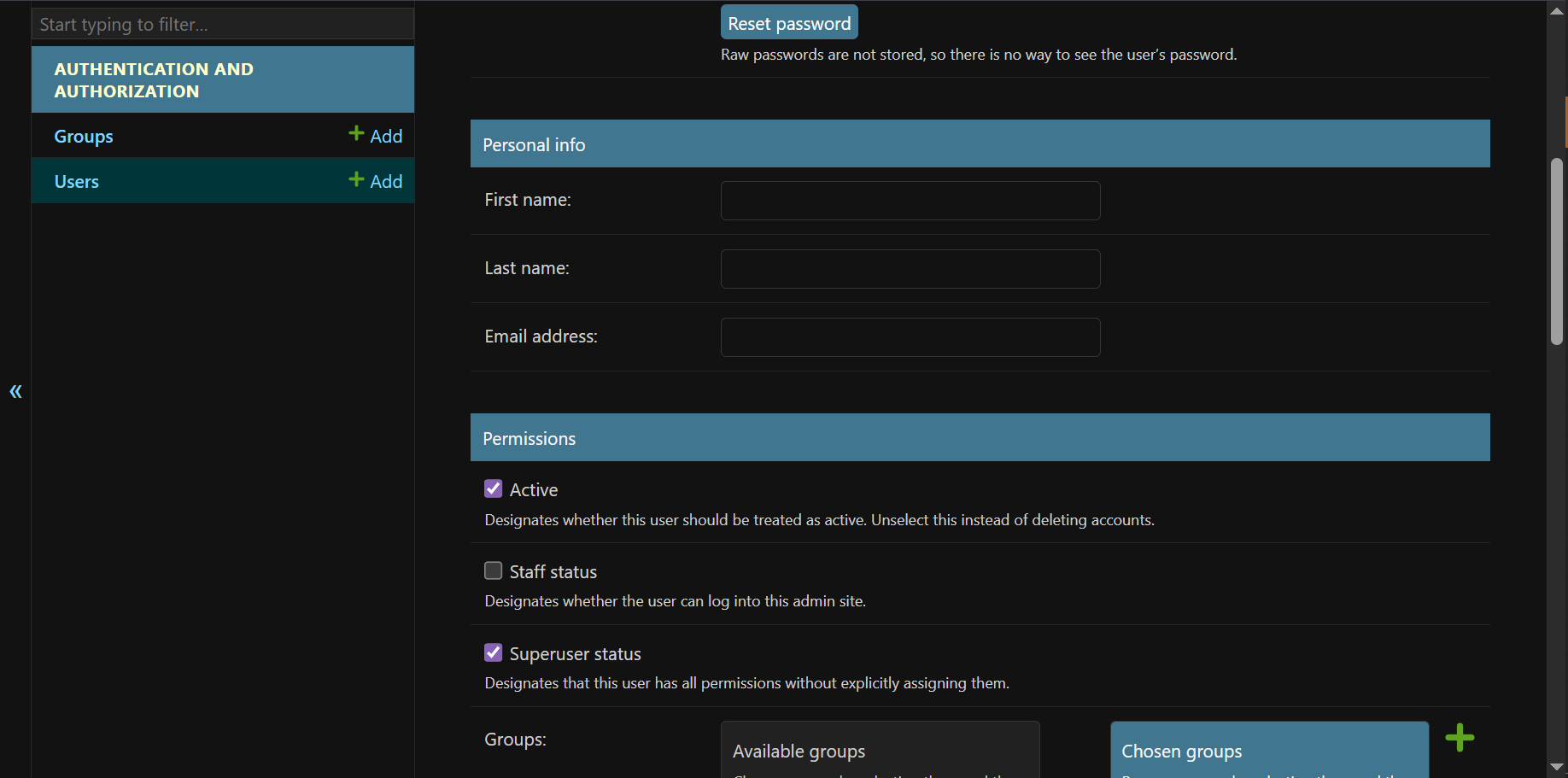
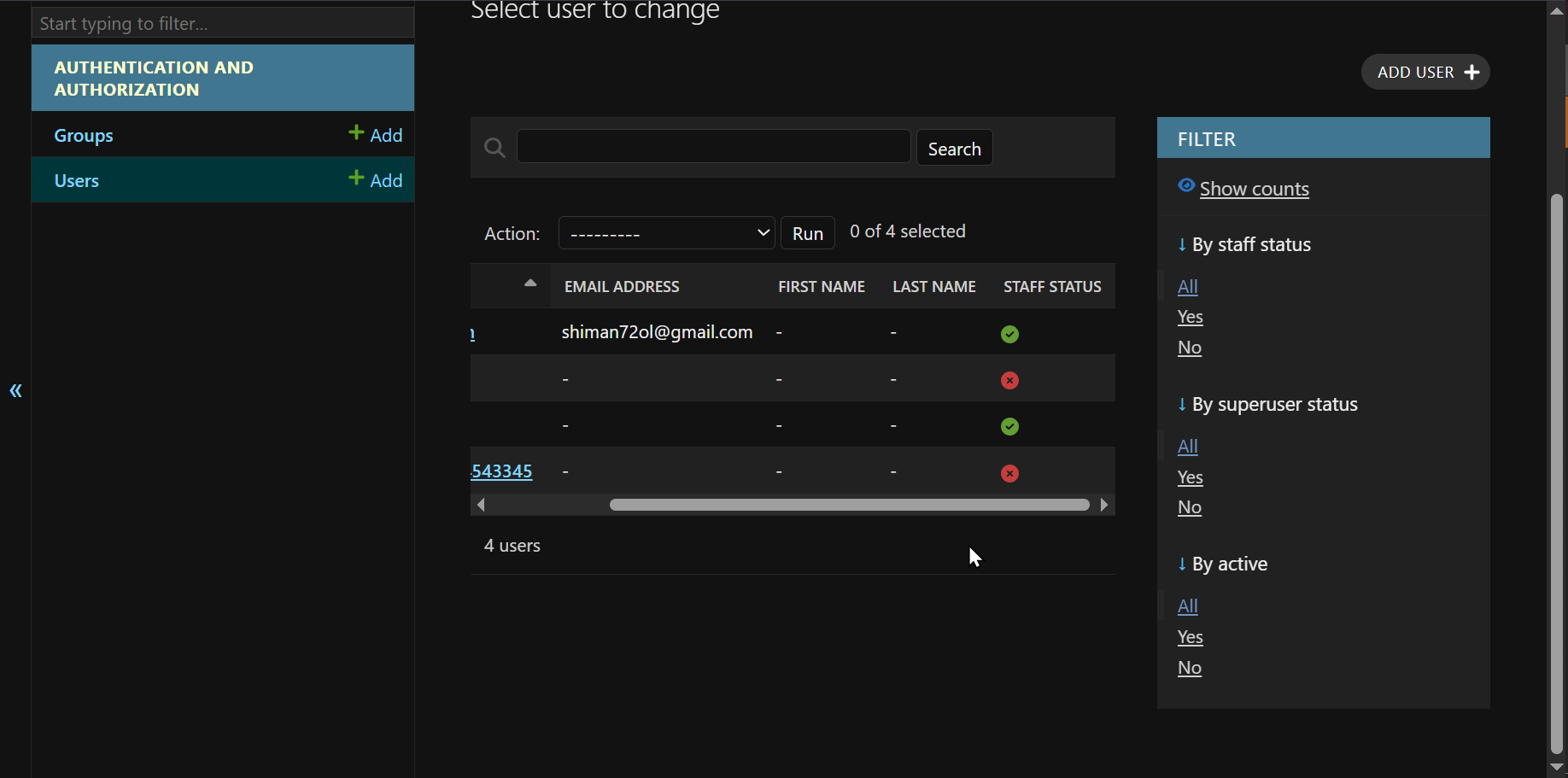
python3 manage.py runserver

# и в браузере пройти по адресу <http://127.0.0.1:8000/admin/>.





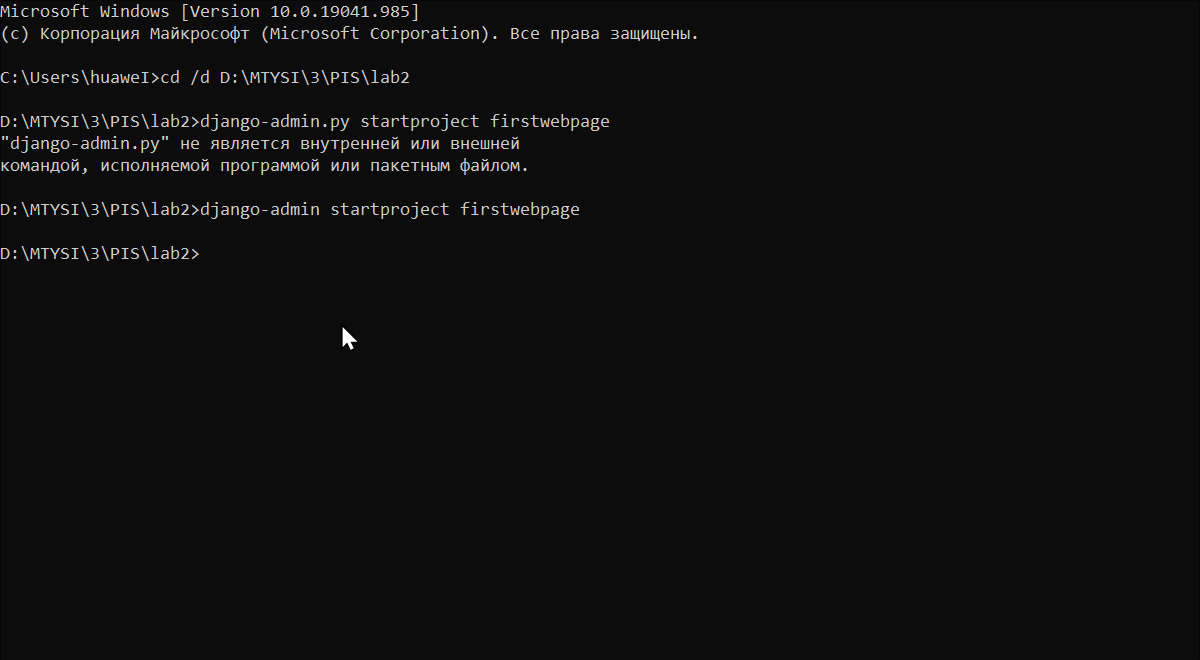
После этого должно появиться окно входа, куда необходимо ввести данные, которые вы указали при создании суперпользователя. Когда вы успешно войдете в аккаунт суперюзера, перед вами откроется главная страница административной панели, через которую вы можете управлять вашими приложениями, редактируя существующие записи в базе данных или генерируя новые.

* Изучите интерфейс административного приложения django;
* Через интерфейс административного приложения создайте нового пользователя с правами суперпользователя;
* 
* Через интерфейс административного приложения создайте нового пользователя без прав суперпользователя;
* 
* Через интерфейс административного приложения «забаньте» одного из пользователей (сделайте пользователя «неактивным»);
* 
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).

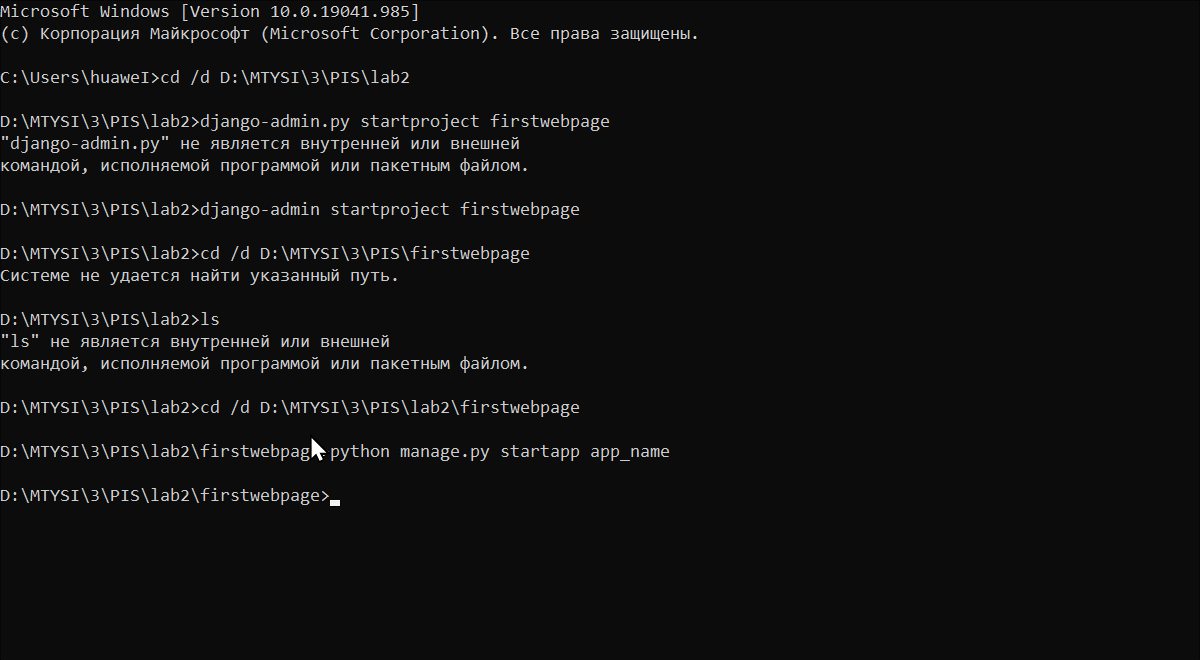
# **Лабораторная работа №2**

*Создание web-страницы с текстом*

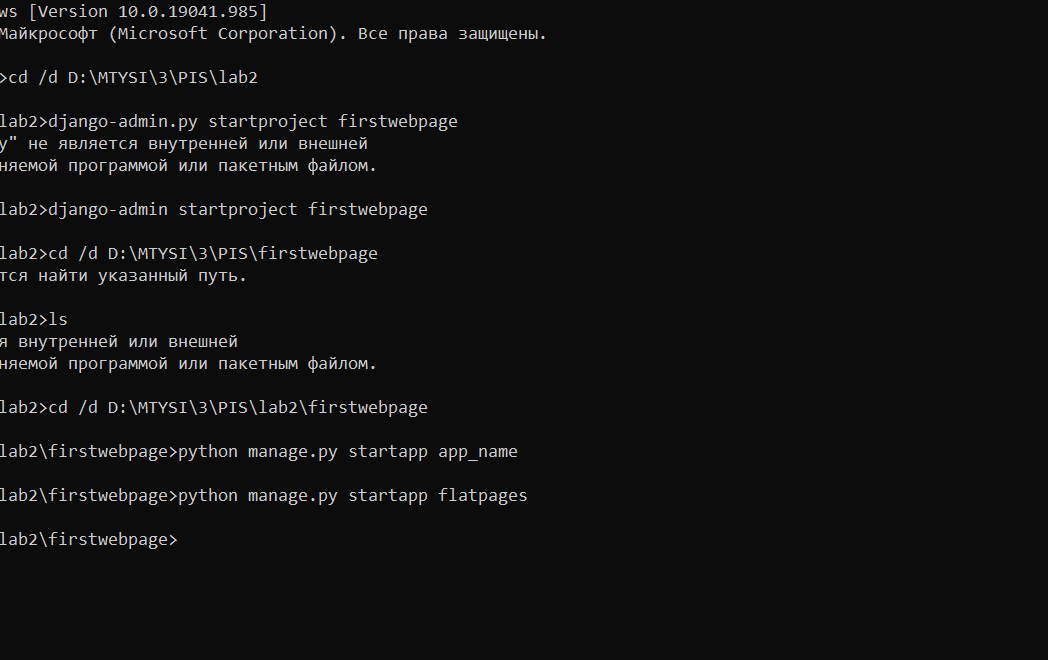
Для выполнения данной лабораторной работы создайте директорию lab2, а в ней, с помощью командной строки и команды: django-admin.py startproject firstwebpage создайте новый проект **firstwebpage**.



Каждый проект в Django – это набор нескольких приложений, каждое из которых выполняет определенную задачу. Чтобы добавить в наш проект новое приложение, необходимо перейдите в созданную директорию **firstwebpage** и выполнить команду: python manage.py startapp app\_name, где app\_name – это имя подключаемого приложения.



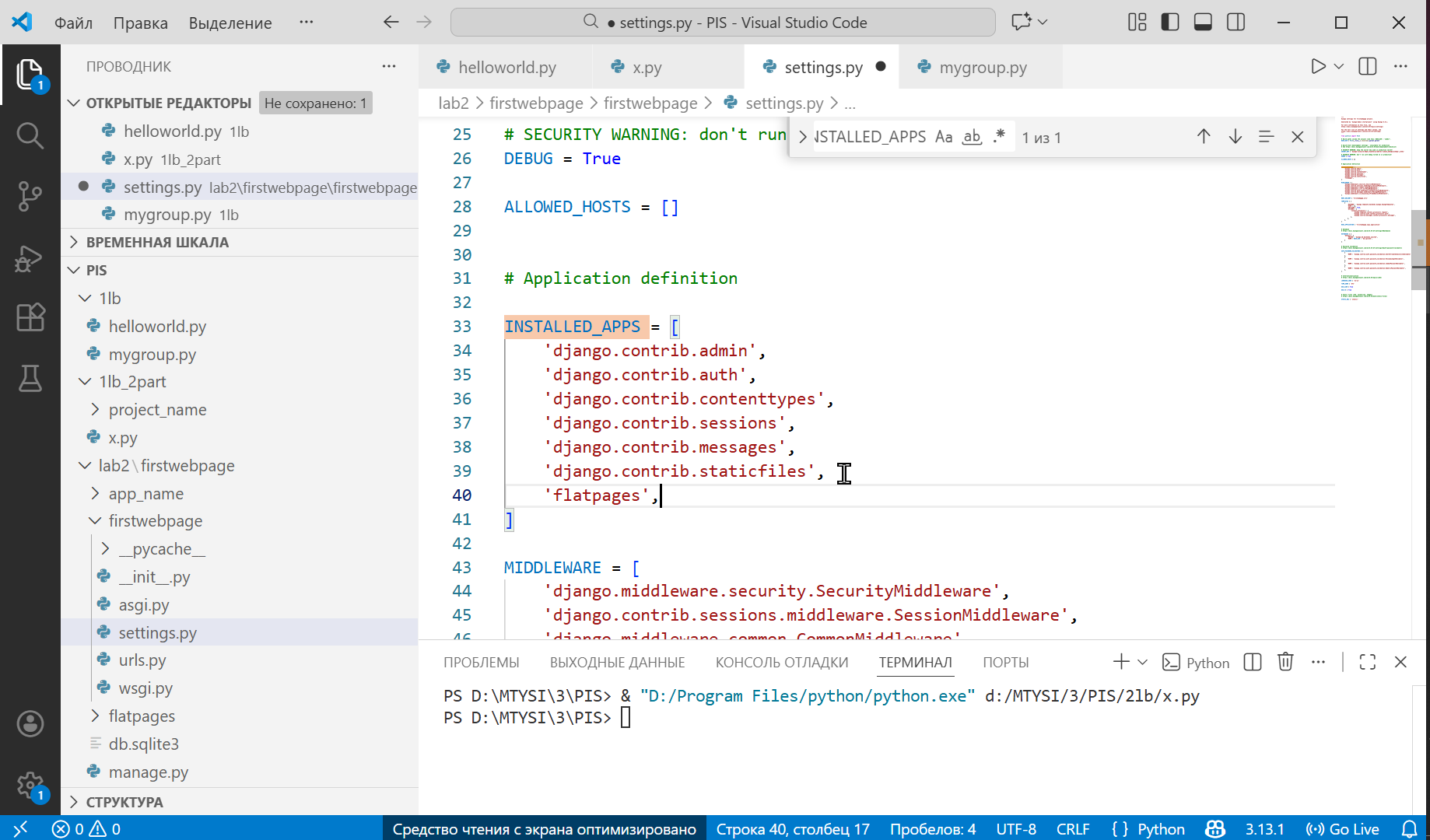
Вам необходимо подключить приложение **flatpages**. Это приложение позволяет вам управлять статическими страницами через интерфейс администратора Django и указывать шаблоны для таких страниц с помощью шаблонной системы Django.



В начале работы над новым проектом необходимо задать базовые настройки, например, задать имя базы данных и создать ее таблицы (см. Лабораторную работу №1), а также добавить нужные приложения в проект.



Чтобы добавить приложение **flatpages** в ваш проект, откройте файл settings.py, найдите кортеж INSTALLED\_APPS и добавьте в конец элемента строку 'flatpages'.



Перед началом работы с файлом **urls.py** необходимо импортировать views нашего приложения:

from flatpages import views

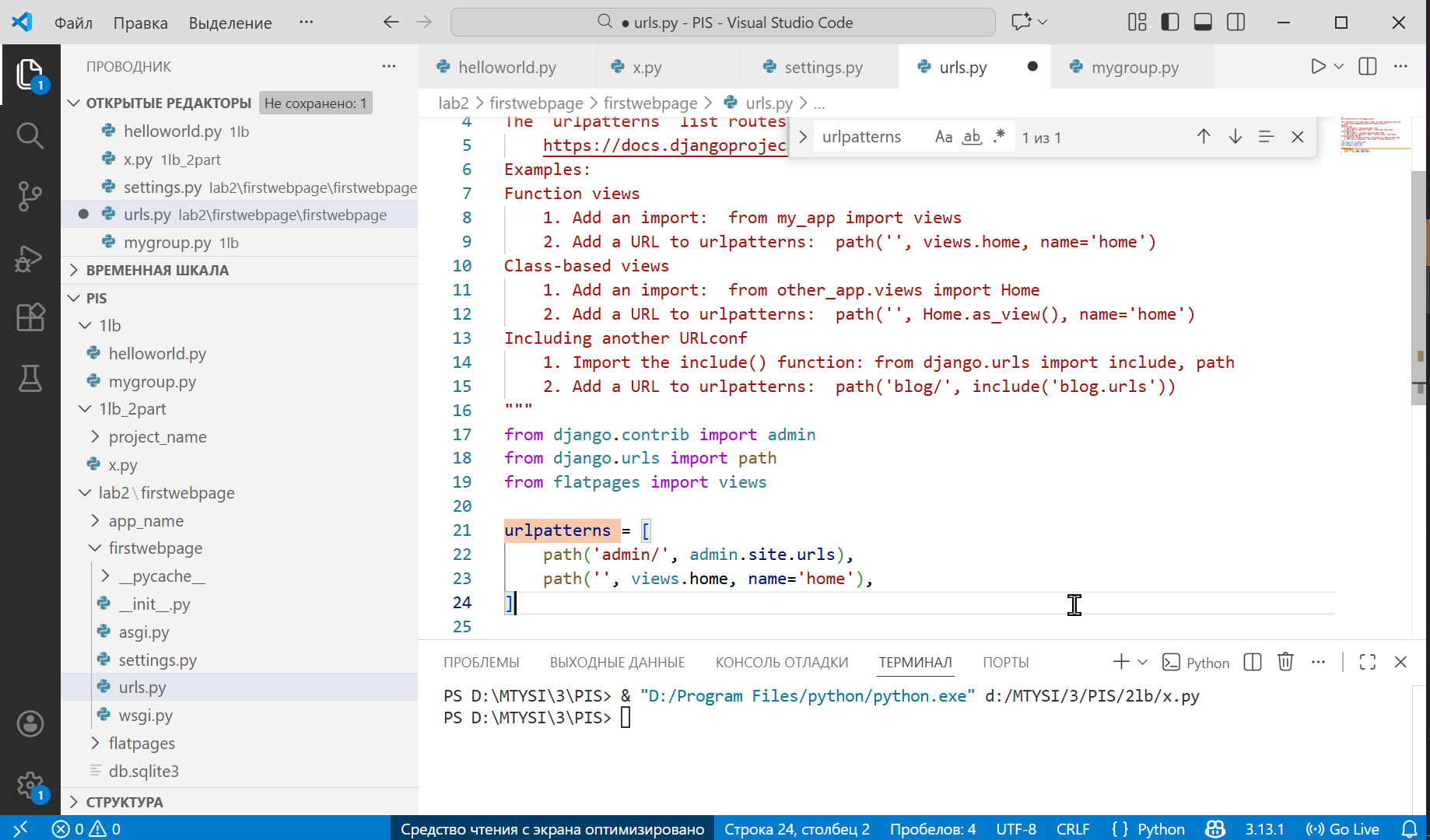
Для будущей страницы создайте новый адрес в файле **urls.py**:

urlpatterns = [

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

path('', views.home, name='home'),

]



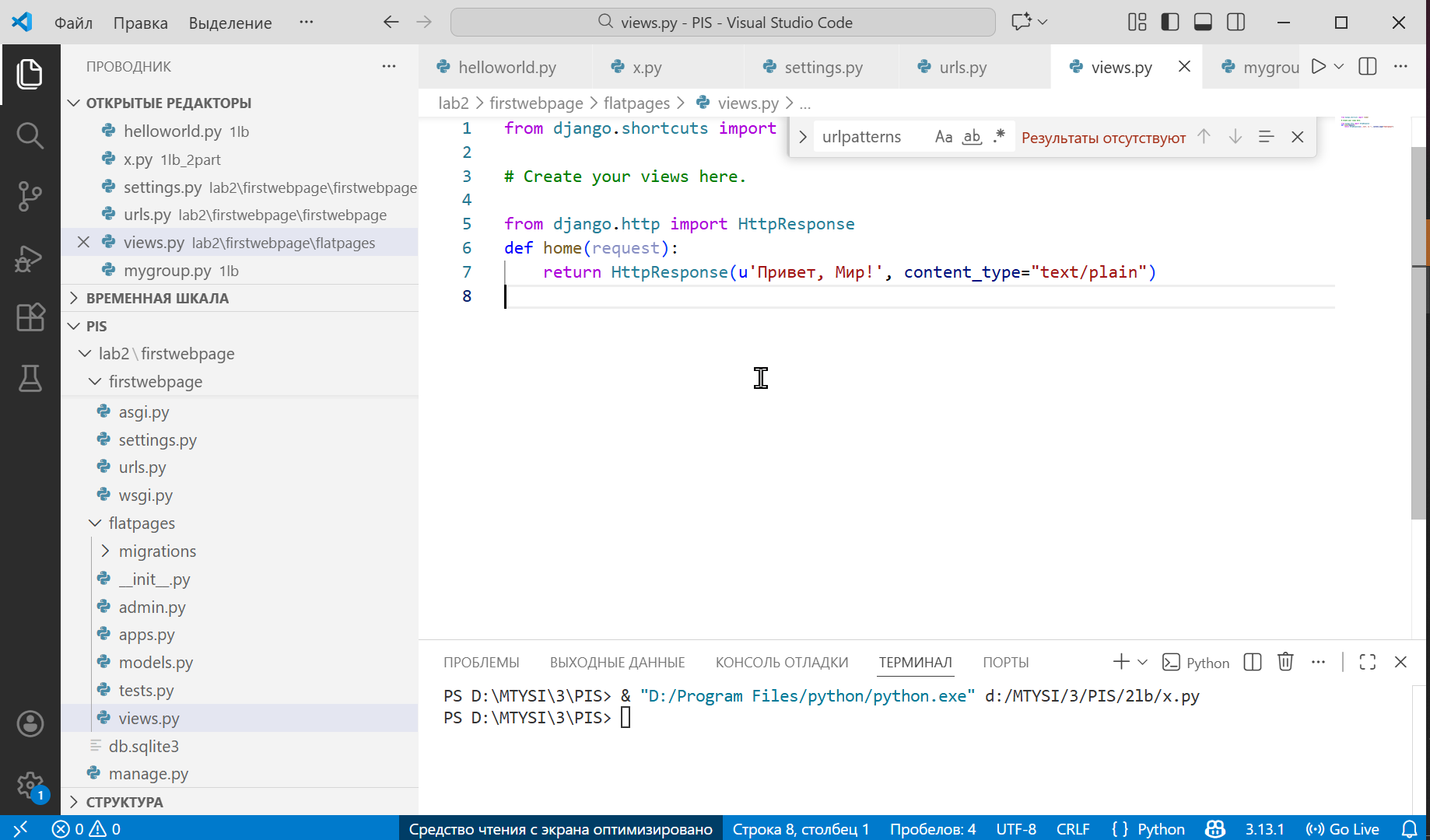
Второй аргумент, передаваемы в функцию, указывает на то, что функция представления, которая будет обрабатывать запросы по этому адресу, находится в файле **views** и называется **home**.

Для того, чтобы в будущем при обращении генерировался ответ, необходимо создать функцию home в файле **views.py** в директории **flatpages**.

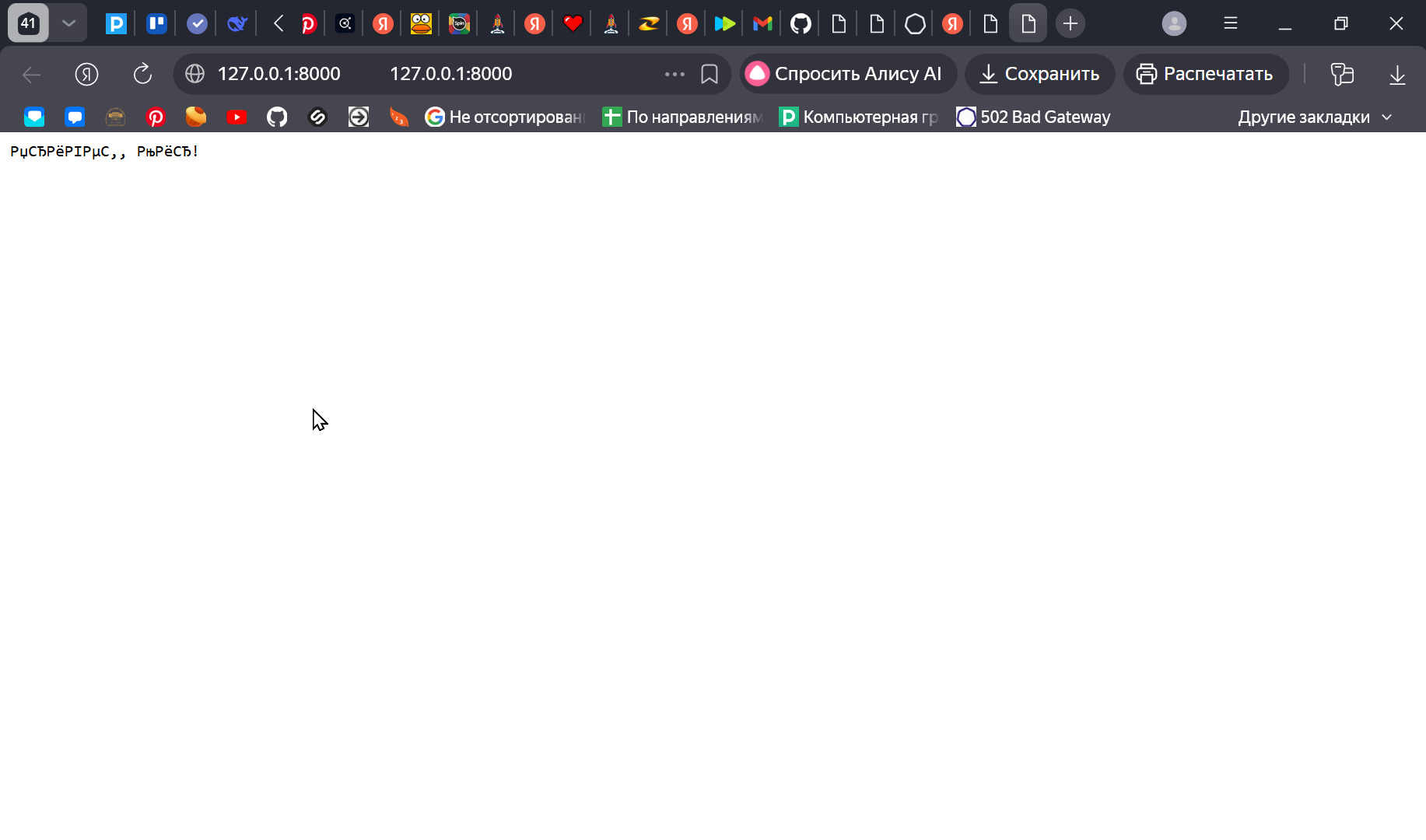
from django.http import HttpResponse

def home(request):

return HttpResponse(u'Привет, Мир!', content\_type="text/plain")

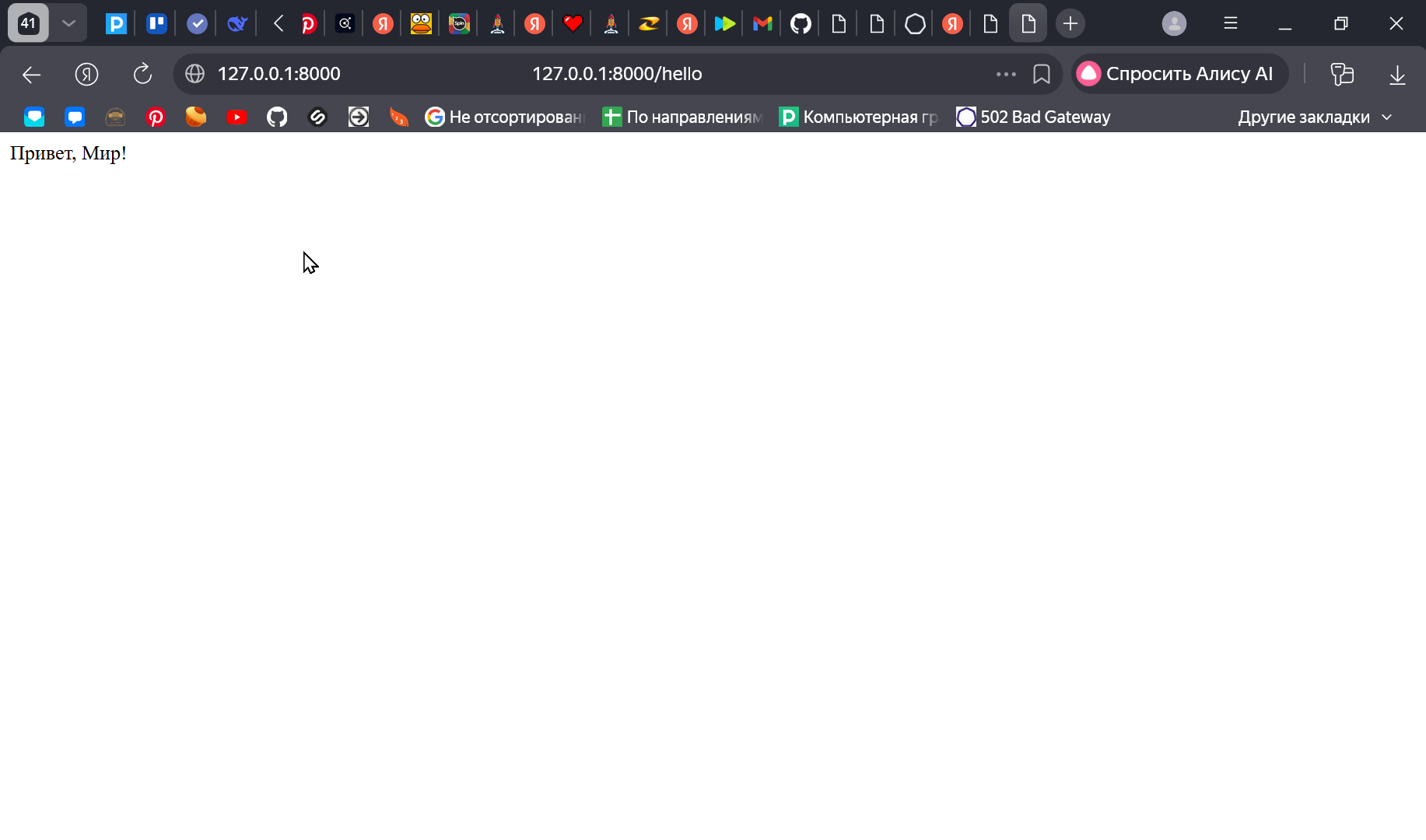


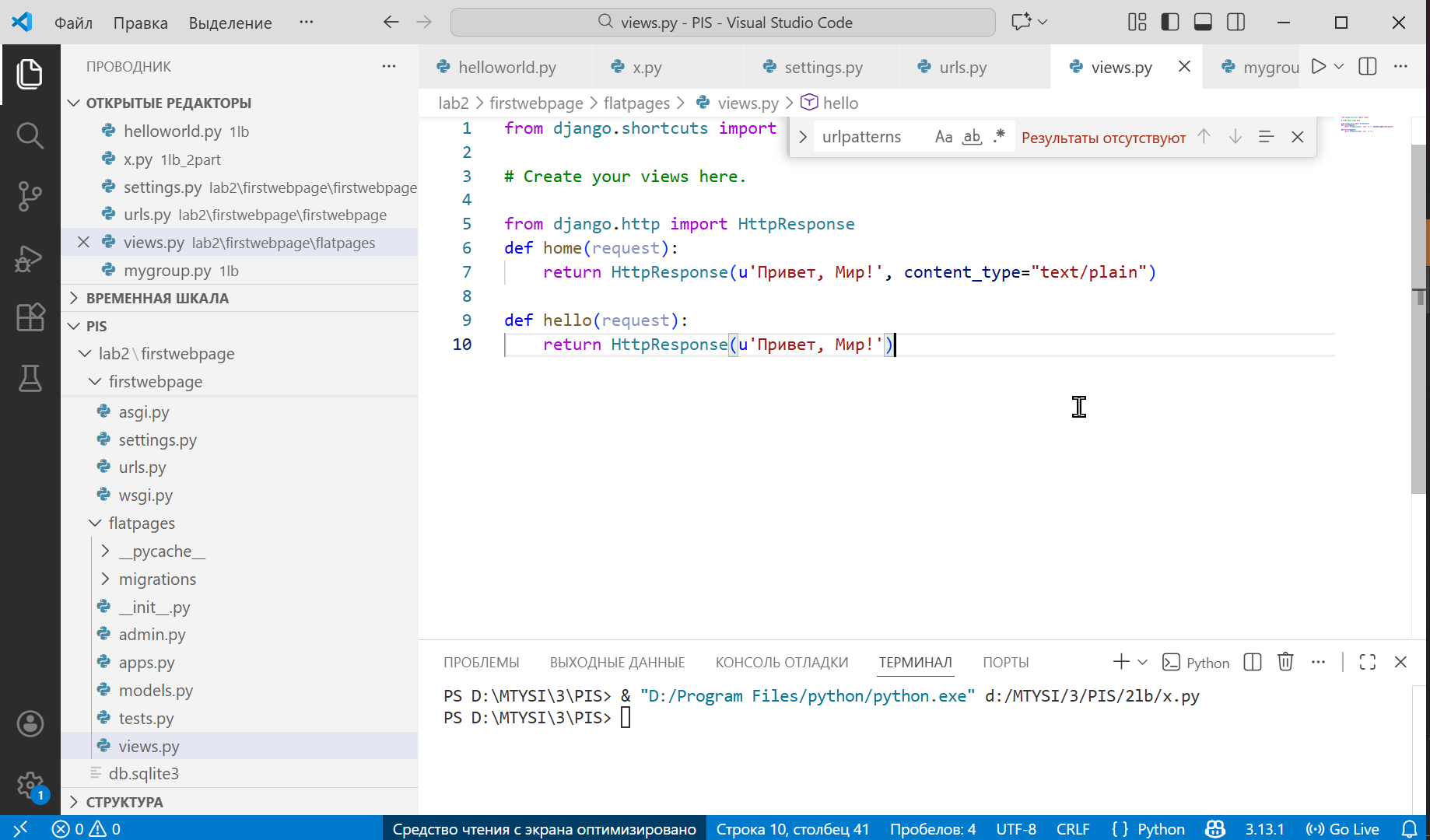
Сначала происходит импортирование специального класса HttpResponse, любая функция представления должна возвращать переменные, где хранятся представители именно этого класса или класса, наследующего HttpResponse. В данном случае функция представления **home** является типичными примером представлений в Django – она принимает на входе объект запроса и возвращает на выходе объект ответа. В нашем случае в теле объекта ответа будет находится простая строка 'Привет, Мир!', а в качестве типа ответа указан простой текст. Теперь можно зайти на страницу по адресу http://127.0.0.1:8000/ и посмотреть результат.



**Задание:**

* Сделайте так, чтобы по адресу <http://127.0.0.1:8000/hello/> возвращался тот же самый текст;



* Уберите указание типа возвращаемого ответа (если классу HttpResponse напрямую не указать тип ответа, то будет выставлено значение по умолчанию). Сравните полученные результаты.
* 

*Работа с шаблонами в Django*

Шаблон в Django представляет собой строку текста, предназначенную для отделения представления документа от его данных. В шаблоне могут встречаться маркеры и простые логические конструкции (теги), управляющие отображением документа. Обычно шаблоны применяются для создания HTML-разметки, но в Django они позволяют генерировать документы в любом текстовом формате. Шаблон – это основа будущей HTML-разметки, которая должна быть заполнена теми данными, которые будут переданы в шаблон. Данные, которые необходимо внести в шаблон, называются *контекстом*, а процесс, когда данные вносятся в шаблон, называется *рендерингом*.



Создайте папку **templates** в директории **flatpages**. Затем, в папке **templates** создайте файл **index.html** со следующим кодом:

<!DOCTYPE html>

<html>

<head>

<title>Привет, Мир!</title>

</head>

<body>

<h1>Привет, Мир!</h1>

<h2>Это учебный сайт, с его помощью будут изучены технологии

python/django, html/css.</h2>

<h3>Как видите, здесь используются заголовки различных

уровней.</h3>

<p>Здесь есть маркированный список:</p>

<h4>

<ul>

<li>Элемент 1;</li>

<li>элемент 2;</li>

<li>элемент 3;</li>

<li>последний элемент.</li>

</ul>

</h4>

<p>И нумерованный список:</p>

<h4>

<ol>

<li>Элемент 1;</li>

<li>элемент 2;</li>

<li>элемент 3;</li>

<li>последний элемент.</li>

</ol>

</h4>

<p>И даже таблица:</p>

<table style="border: none">

<thead>

<tr>

<th>Столбик 1</th>

<th>Столбик 2</th>

<th>Столбик 3</th>

<th>Столбик 4</th>

</tr>

</thead>

<tr>

<td>Строка 1 Столбец 1</td>

<td>Строка 1 Столбец 2</td>

<td>Строка 1 Столбец 3</td>

<td>Строка 1 Столбец 4</td>

</tr>

<tr>

<td>Строка 2 Столбец 1</td>

<td>Строка 2 Столбец 2</td>

<td>Строка 2 Столбец 3</td>

<td>Строка 2 Столбец 4</td>

</tr>

<tr>

<td>Строка 3 Столбец 1</td>

<td>Строка 3 Столбец 2</td>

<td>Строка 3 Столбец 3</td>

<td>Строка 3 Столбец 4</td>

</tr>

<tr>

<td>Строка 4 Столбец 1</td>

<td>Строка 4 Столбец 2</td>

<td>Строка 4 Столбец 3</td>

<td>Строка 4 Столбец 4</td>

</tr>

<tr>

<td>Строка 5 Столбец 1</td>

<td>Строка 5 Столбец 2</td>

<td>Строка 5 Столбец 3</td>

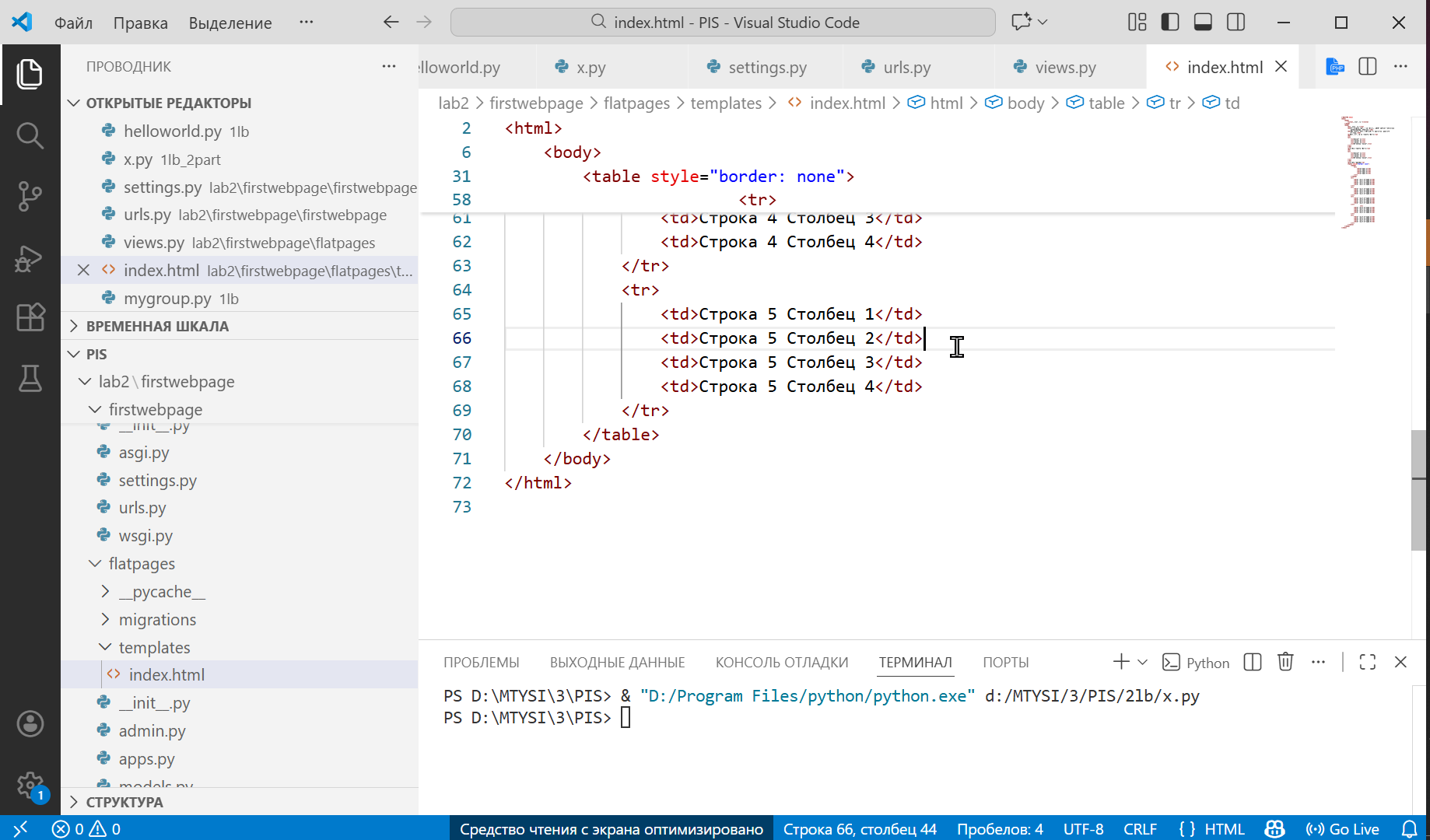
<td>Строка 5 Столбец 4</td>

</tr>

</table>

</body>

</html>



Теперь созданный html-файл необходимо подключить к функции-представлению home, чтобы вместо простого текстового ответа приходил html-документ. Для этого в файле views.py, который находится в директории flatpages, добавьте операции импортирования:

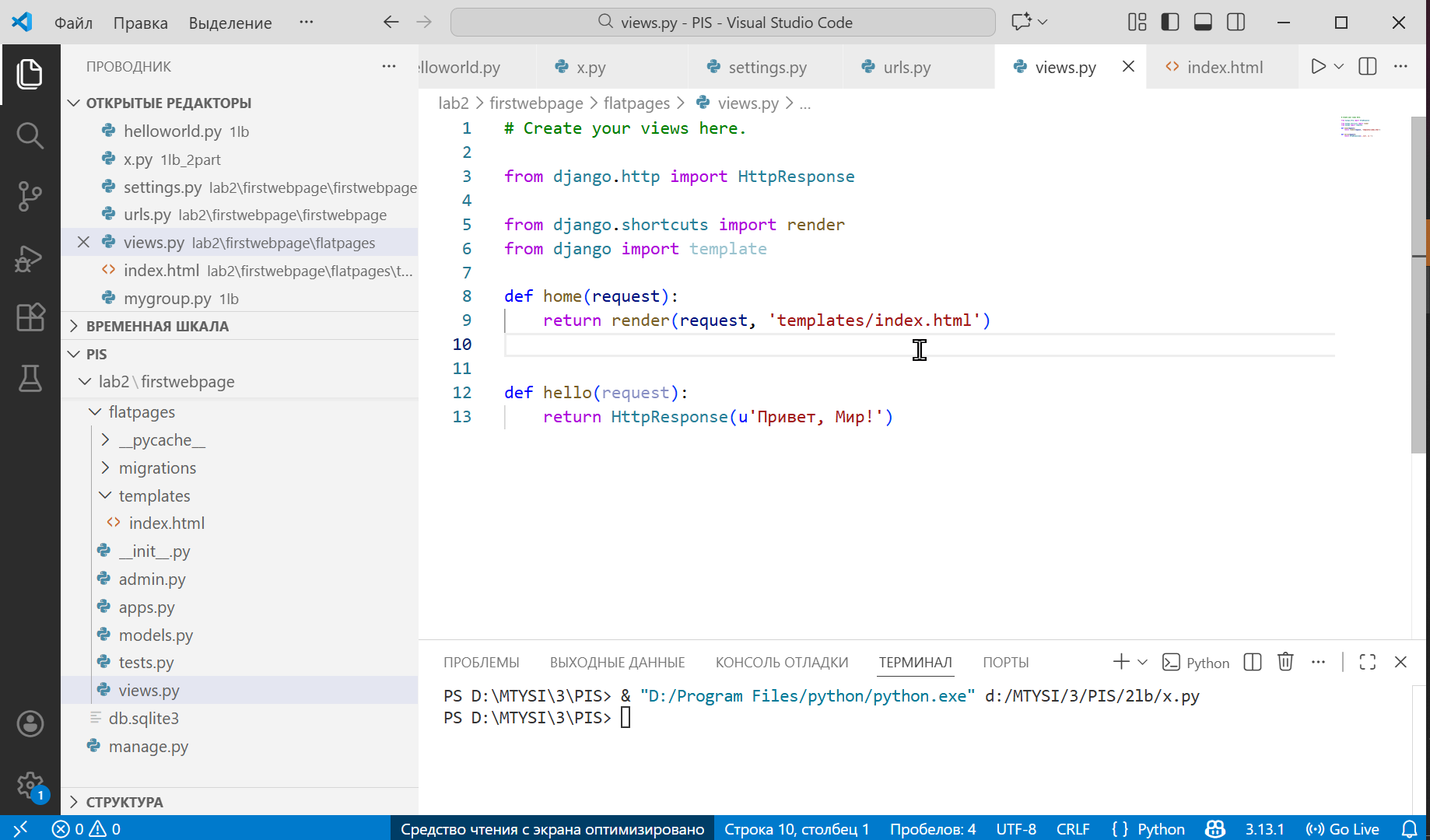
from django.shortcuts import render

from django import template

А функцию-представление home изменим следующим образом:

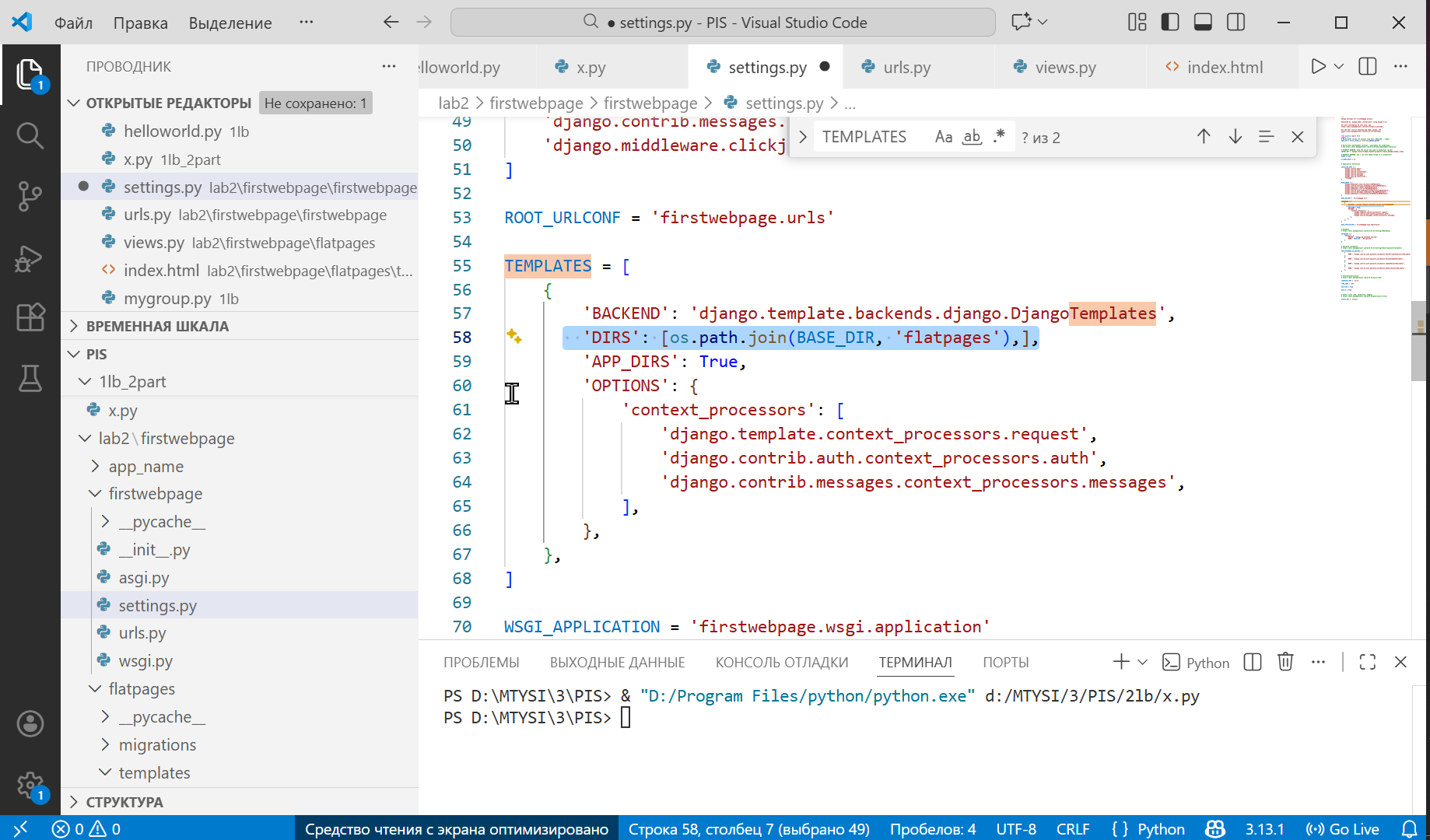
def home(request):

return render(request, 'templates/index.html')



Также, для того, чтобы файл **index.html** был найден в директории **templates**, необходимо в файле **settings.py** изменить поле **DIRS** кортеже **TEMPLATES**. Поле **DIRS** должно содержать адрес директории, в которой располагается файл **index.html**, например:

'DIRS': [os.path.join(BASE\_DIR, 'flatpages'),],



Представление и шаблон готовы. Для того, чтобы посмотреть изменения, перезагрузите страницу с адресом <http://127.0.0.1:8000/>, если сервер запущен.

*Пример:*

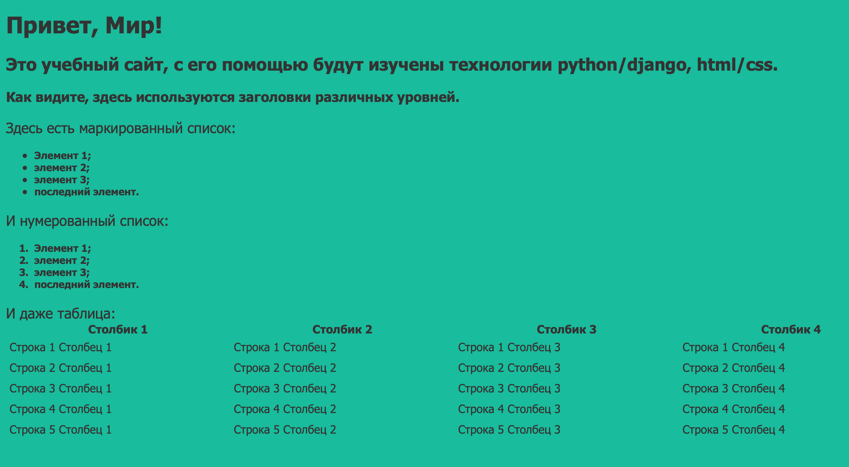
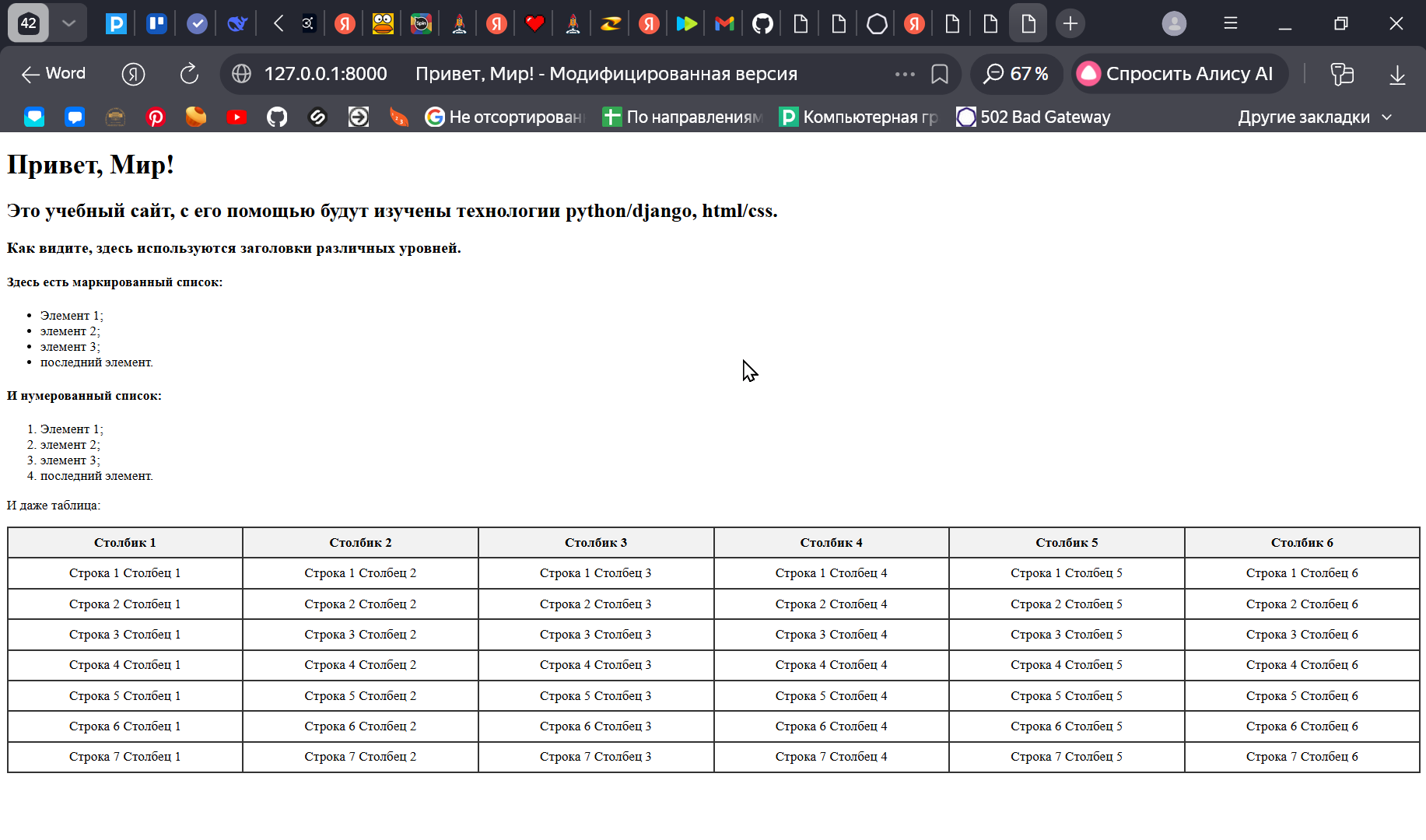
**

Рисунок 3. Пример отображения страницы

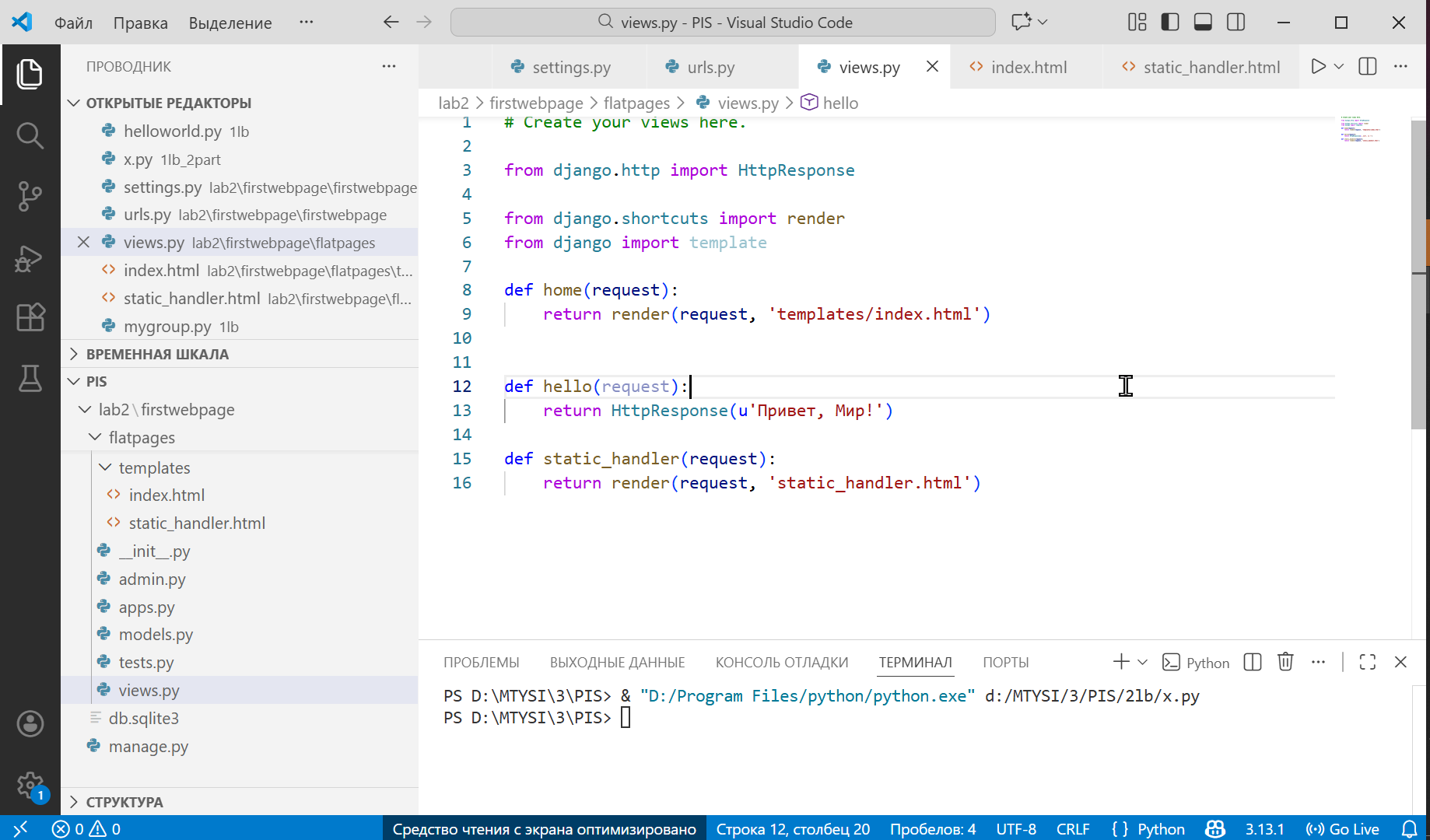


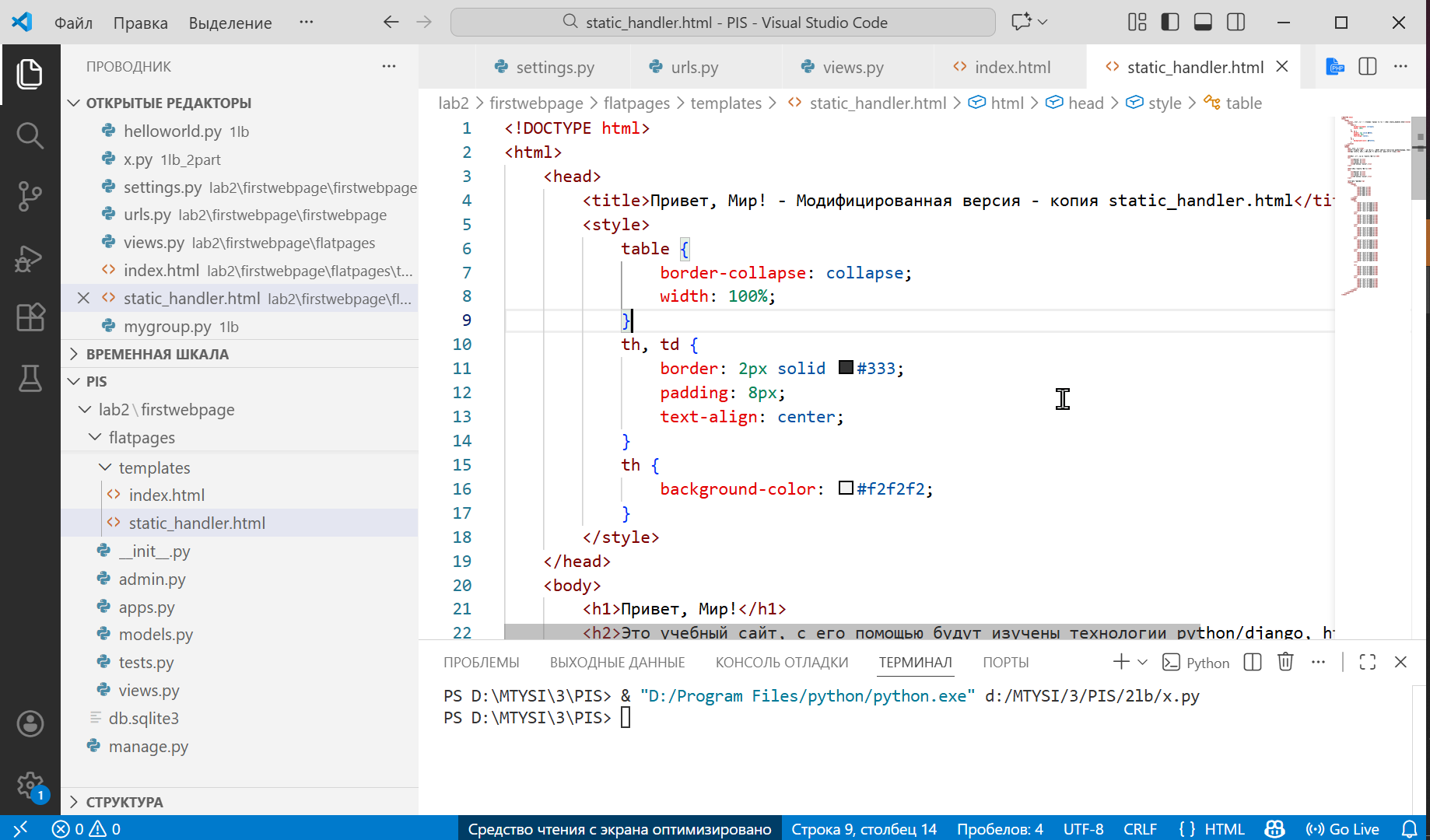
**Задание:**

* Добавьте к созданной таблице 2 строки и 2 столбца;
* Добавьте границы для таблицы;
* Сделайте заголовки списков (нумерованного и маркированного) подзаголовками четвертого уровня;



* Создайте абсолютно идентичный шаблон, изменив только название на «static\_handler.html». В следующих заданиях при выполнении этой лабораторной работы изменяйте именно этот файл.





*Настройка обработки статичных файлов для Django*

Django-разработчики в основном работают с динамической частью приложения – представлениями и шаблонами, которые чаще всего изменяют свое содержимое при каждом запросе (например, страница профиля /profile/ будет у каждого пользователя разная, хотя каркас у всех будет одинаковым). Но web-приложения содержат и другую часть: статические файлы (изображения, CSS, Javascript и др.), которые не требуют никакой программной обработки. Для них нет потребности в рендеринге, они не зависят от содержимого базы данных. При каждом запросе к такому веб-серверу достаточно просто вернуть их прямо такими, какими их сохранили в последний раз.

Для того, чтобы придать документу стиль в соответствии с макетом, будем использовать CSS (формальный язык описания внешнего вида документа, написанного с использованием языка разметки). Создайте папку **static** в директории **flatpages**, а в ней файл **index.css** с кодом:

body {

background: #1abc9c;

font-family: Tahoma, Arial, sans-serif;

color: #333;

}

table {

border-collapse: collapse;

}

p, h4 {

font-size: 20px;

margin-bottom: 0;

}

h4 {

font-size: 14px;

}

ul, ol {

margin: 0;

}

table tr td {

padding: 5px;

}

table {

width: 100%;

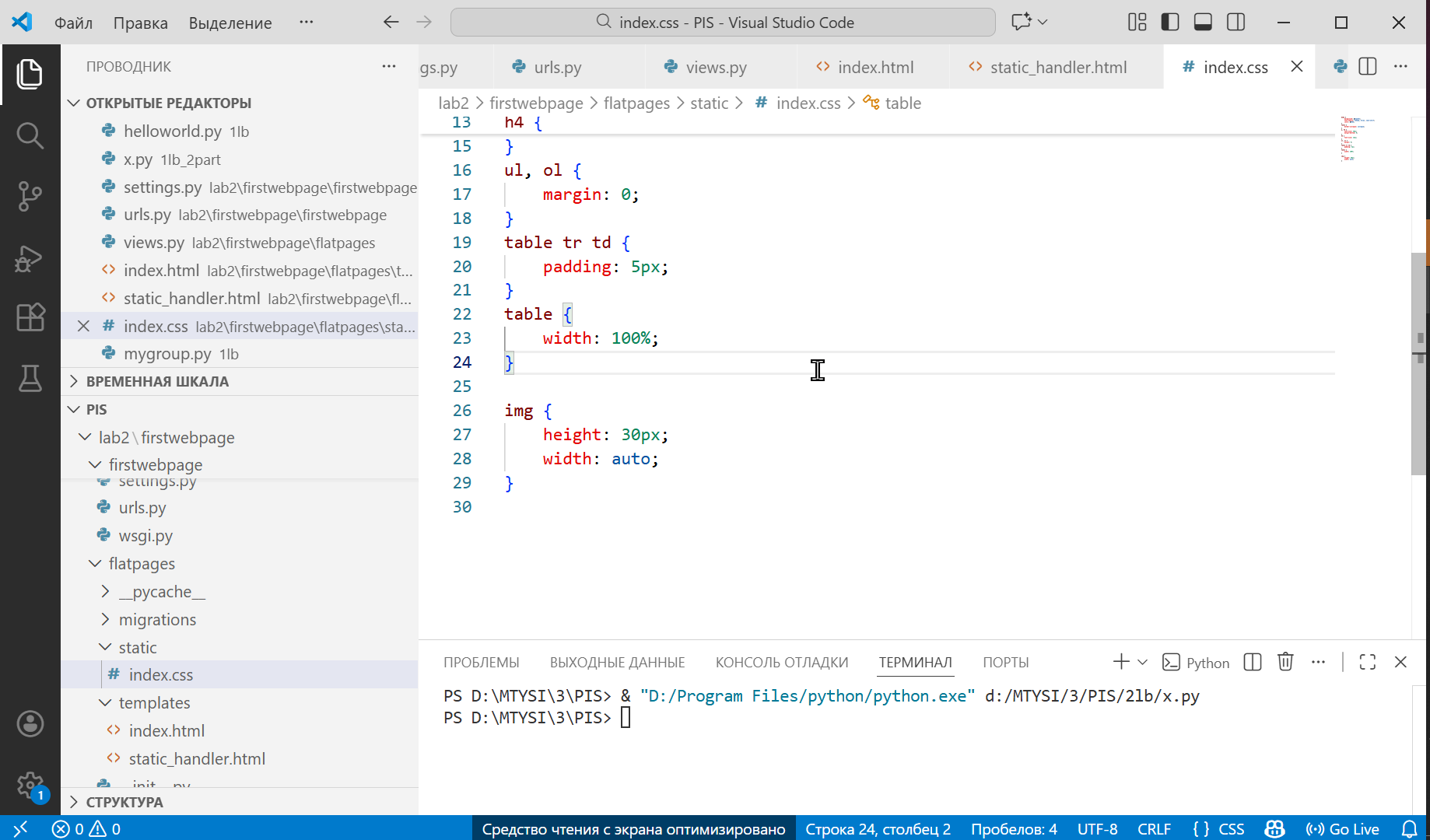
}

img {

height: 30px;

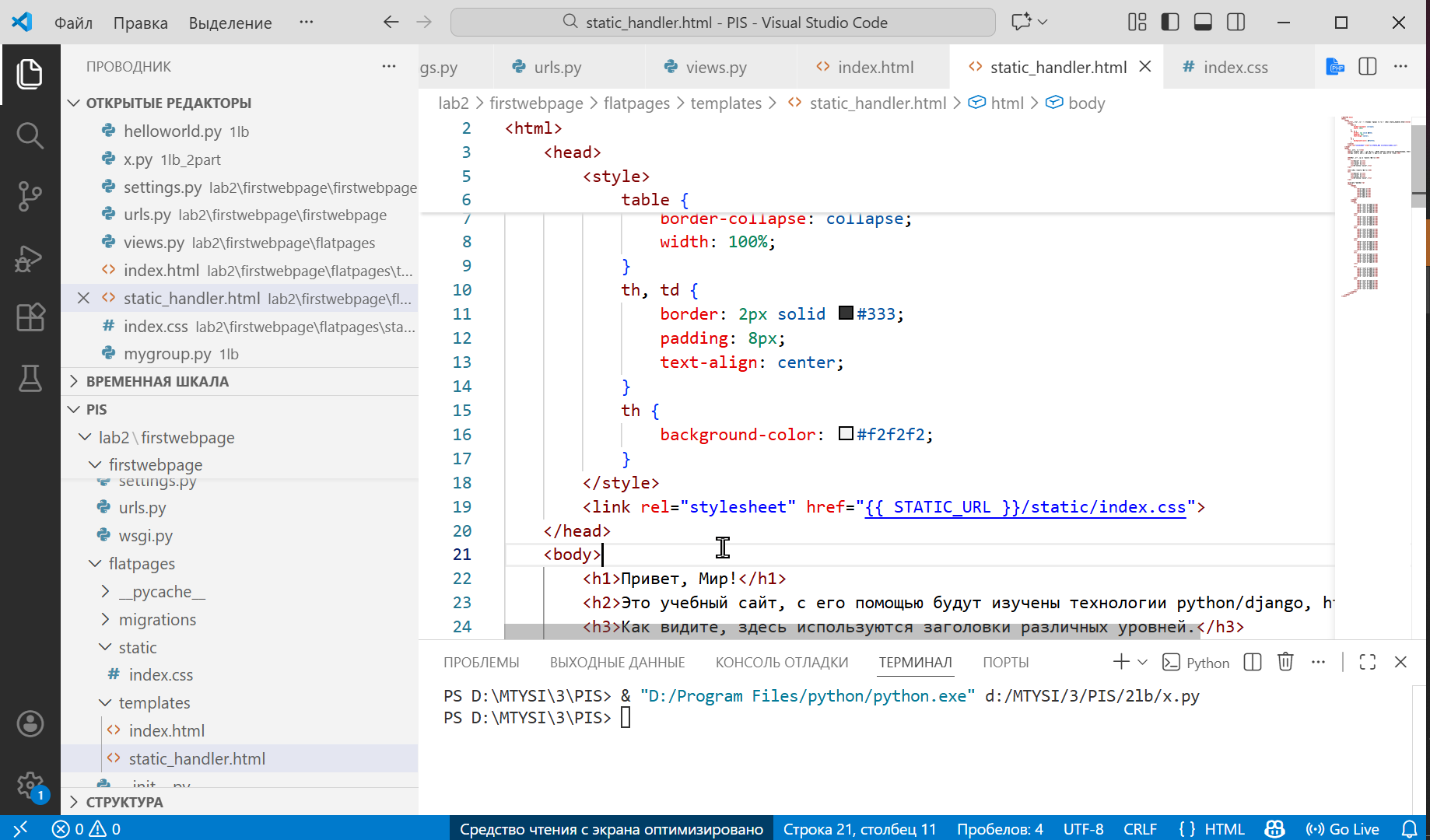
width: auto;

}



После этого в файл страницы **static\_handler.html** во внутрь тега **<head>** вставьте тег подключения css-скрипта:

<link rel="stylesheet" href="{{ STATIC\_URL }}/static/index.css">



STATIC\_URL – переменная, объявленная в файле **settings.py**.

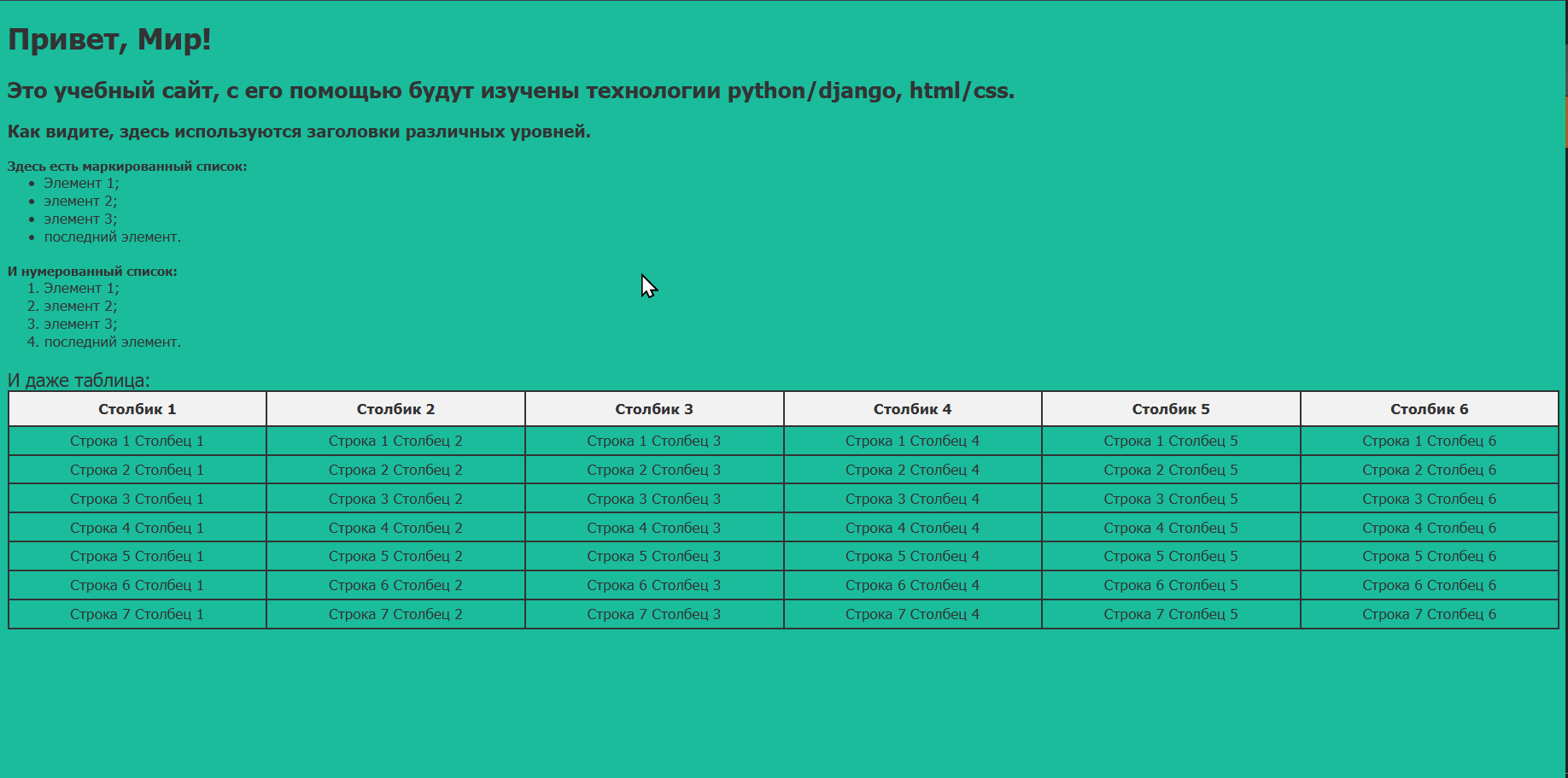
Свойство **background** задает фоновый цвет для элемента, в данном случае цвет задан в формате HEX (то есть три идущих подряд двузначных шестнадцатеричных числа).

Свойство **font-family** задает семейство шрифтов, которые будут применены к элементам.

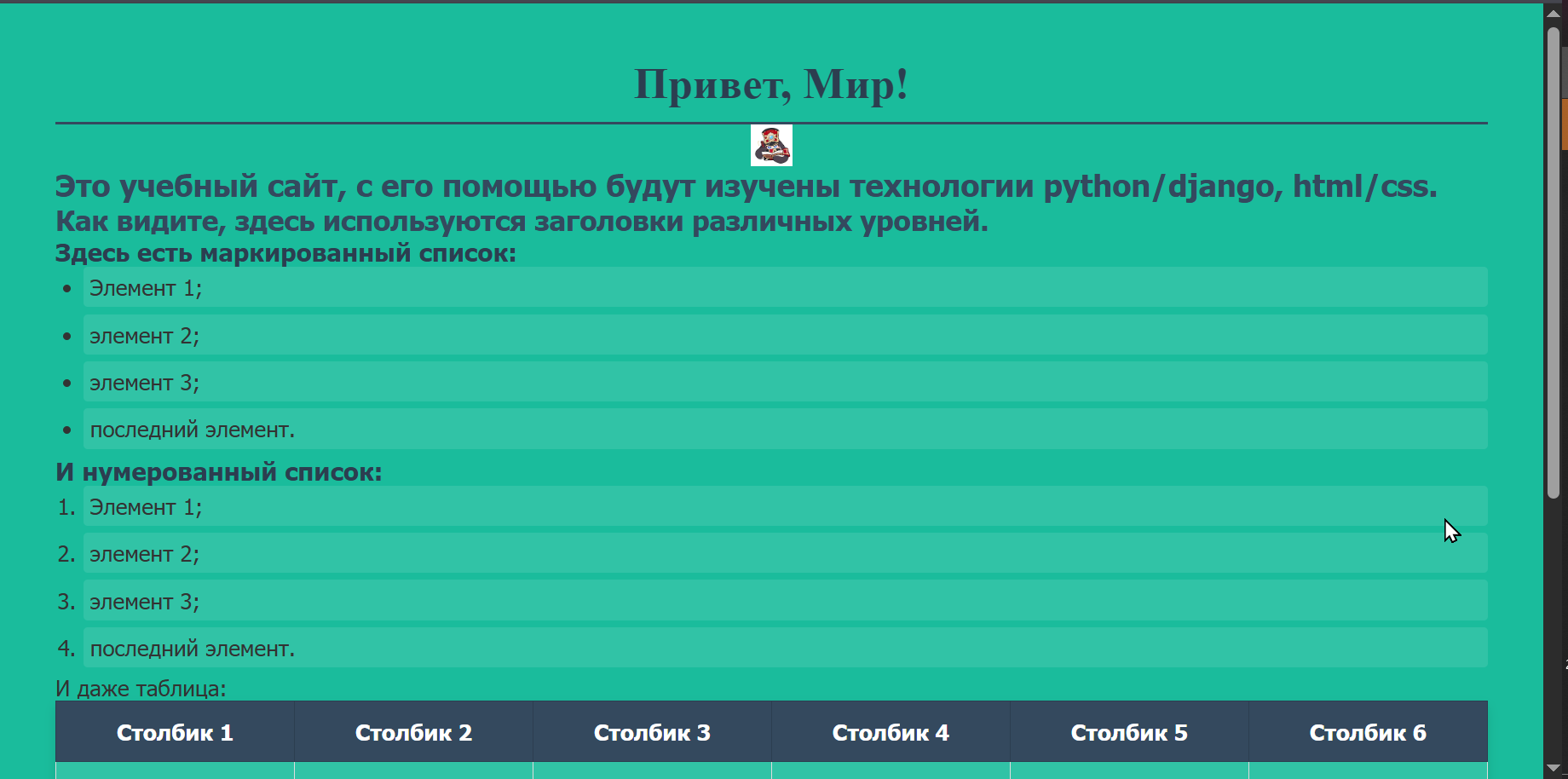
Свойство **color** задает цвет шрифта.

Свойства border-collapse, font-size, margin и padding и дальнейшие применяемые свойства остаются на самостоятельное изучение.

Когда перед фигурными скобками указано несколько элементов через запятую, значит стиль применится ко всем перечисленным элементам.



**Задание:**

* Установите для заголовка первого уровня шрифт с засечками;
* Добавьте картинку и сделайте ее высотой 30px;
* Измените размер шрифта для подзаголовков четвертого уровня;
* Сделайте ширину таблицы на 100% экрана;
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).
* 

# **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 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.

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

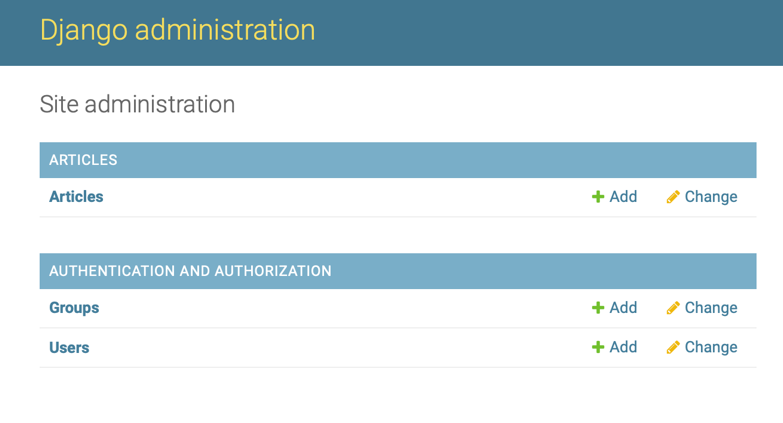


Рисунок 4. Административная панель

**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: <img src="{{ STATIC\_URL }}img/logo.png" />

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.).

# **Лабораторная работа № 4**

*Создание страницы определенной записи*

Вы уже создавали страницу для всех постов, теперь создадим страницу отдельного поста. Для начала необходимо будет создать адрес будущей страницы, затем написать функцию-представление для нее и реализовать шаблон. Адрес страницы для одной записи обязательно должен нести в себе какое-то ключевое слово (ключ), чтобы можно было среди всей таблицы базы данных найти интересующий пользователя экземпляр. Для этого можно использовать **id** (уникальный идентификатор)сообщений, которые устанавливаются автоматически при создании каждого отдельного объекта сообщений. В итоге адрес страницы для статьи с идентификатором 5 должен выглядеть так: <http://127.0.0.1:8000/article/5/>.

В Django довольно часто используются регулярные выражения (формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов), они нужны для того, чтобы было легче извлекать данные из адресов, по которым приходят запросы на сервер.

Для начала выполнения лабораторной работы, в директории **lab4** создайте новый проект **blog**, проведите первоначальную настройку и перенесите приложение articles из предыдущей работы (скопируйте папку в этот проект, добавьте нужные данные в INSTALLED\_APP и настройте файл **urls.py**).

Для того, чтобы можно было бы прочитать значение идентификатора (id) отдельного поста, укажите в качестве адреса следующее регулярное выражение (в файле urls.py):

urlpatterns = [

path(r'^article/(?P<article\_id>\d+)$', views.get\_article, name='get\_article')

]

В качестве первого аргумента передается регулярное выражение, которое возвращает именованную группу article\_id (именованная группа – это переменные внутри регулярного выражения, которые имеют следующий синтаксис: “(?P<name> ...)”, где name вы должны указать сами, а вместо троеточия подставить нужный вам шаблон). Обратите внимание, что именованная группа идет после символов “article/”, и состоит она только из цифр, продолжающихся до завершения адреса. Соответственно, если хоть одно из условий не подходит (например, адрес начинается с “artikle/” вместо “article/”, или та комбинация, что идет после “article/” состоит не только из цифр, но и из латинских букв), то регулярное выражение не подойдет и представление views.get\_article не запустится.

После того, как в регулярном выражении вы объявили именованную группу, в представление можно добавлять именованный аргумент, куда и будет передано значение из адреса, по которому перешел пользователь. Создайте в файле **views.py** из директории **articles** помимо уже написанной в предыдущей работе функции archieve ещё и функцию get\_article:

from django.http import Http404

def get\_article(request, article\_id):

try:

post = Article.objects.get(id=article\_id)

return render(request, 'article.html', {"post": post})

except Article.DoesNotExist:

raise Http404

Здесь использовался синтаксис обработки исключений, который помогает вручную обрабатывать возникающие ошибки (если ошибка была именно той, что указано после ключевого слова except). Если возникла ошибка Article.DoesNotExist (элемент с указанным id, который искался через функцию get(), отсутствует) возбуждается другая ошибка, системная, которая будет обработана автоматически платформой Django – ошибка Http404. В случае возникновения исключения Http404, Django вернет стандартную страницу ошибки 404 (Page not found).

**Задание:**

* Теперь у определенной записи есть отдельная страница, а значит на странице списка постов каждый заголовок можно сделать ссылкой. Сделайте так, чтобы при клике по названию статьи происходил переход на страницу указанной записи. Для этого достаточно воспользоваться тегом <a>, которому в атрибуте href указать адрес перехода.

*Верстка обеих страниц в соответствии с макетом*

В папке articles/templates/ создайте файл **article.html**. Содержимое его вы можете определить сами, оно будет проще, чем описанная в прошлой лабораторной работе страница вывода всех записей, ведь здесь запись всего одна.

Теперь вам необходимо создать правильные стили. Макет страницы одной записи должен выглядеть как на рисунке 5 (lab4\_item), а макет главной страницы со списком всех постов должен выглядеть как на рисунке 6 (lab4\_main).



Рисунок 5. Макет страницы с одной статьей



Рисунок 6. Макет страницы со списком всех статей

Для того, что придать внешние стили странице, нужно подключить css-файл. Создайте в директории articles/static/css/ файл **article.css**. Теперь подключите этот файл к своим html-шаблонам (articles/templates/archive.html и articles/templates/article.html) так, как вы это делали в лабораторной работе №2.

Для того, чтобы задавать стили, специфичные для одной из страниц (например, только для article.html) тегу <body> каждой страницы задайте уникальный класс. Пусть для **archive.html** это будет класс archive, а для страницы **article.html** будет класс article. Теперь в описании стилей можно указать селектор, например: .article h1 {} Тогда этот стиль применится ко всем заголовкам первого уровня на странице одной записи, но не на странице всех статей (архива).

Задайте общие стили страницы, в теге <body> укажите:

background: #1abc9c;

font-family: Tahoma, Arial, sans-serif;

color: #ffffff;

Чтобы горизонтально выровнять логотип (картинку), достаточно указать определенную ширину и боковые отступы, а также тегам img нужно явно указать, что это блочный элемент:

.header img {

display: block;

width: 318px;

margin-left: auto;

margin-right: auto;

}

Теперь логотип должен располагаться по центру. По такому же принципу необходимо пометить по центру остальной контент этой страницы:

.archive {

width: 960px;

margin-left: auto;

margin-right: auto;

}

После того, как вы сделали названия всех постов ссылками, их цвет стал синий. Исправьте это, указав цвет белым:

.post-title a {

color: #ffffff;

}

Вам необходимо, чтобы имя автора и время создания статьи находились на одном горизонтальном уровне. Имени автора задайте «обтекание слева», при этом указав ширину меньше, чем вся страница, иначе поле «время создания» не поместится справа. Например:

.article-author {

width: 50%;

float: left;

}

Чтобы поле «время создания» находилось в самой правой части экрана, задайте соответствующее выравнивание текста:

.article-created-date {

text-align: right;

}

**Задание:**

* Создайте стили, соответствующие макету для страницы определенной записи. Не забудьте добавить ссылку «Все записи»;
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).

# **Лабораторная работа № 5**

*Создание формы и представления для нового поста*

В директорию **lab5** скопируйте ваш прошлый проект **blog**. В файле **urls.py** создайте ещё один адрес: article/new/, который будет обрабатываться функцией-представлением create\_post. Создайте в файле представлений наряду с функциями archive и get\_article новую функцию create\_post (на которую и ссылается только что написанный url). Эта функция должна будет при первом обращении к странице просто возвращать html-код с формой создания новой записи (html-шаблон для этого вы напишите чуть позже), а в случае, если к функции пользователь обратился во второй и следующий разы, при этом заполнив форму с названием и текстом нового поста, представление должно сохранять отправленные через форму данные в базу данных. Иными словами, сейчас вам необходимо создать представление, которое выполняет ту же роль, что и страница добавления статьи в административном интерфейса (эта страница при запущенном проекте должна быть доступна по ссылке <http://localhost:8000/admin/articles/article/add/>).

Сначала в представлении проверьте, авторизован ли пользователь на сайте, потому что у каждого поста есть поле «автор», которое не может быть корректно установлено, если автор не авторизован. Соответственно, всю работу необходимо проводить, только если данная проверка была успешно пройдена:

def create\_post(request):

if not request.user.is\_anonymous():

# Здесь будет основной код представления

else:

raise Http404

Если пользователь авторизован, то сначала определитесь, что программе необходимо сделать на данный момент: просто вернуть страницу с формой или пользователь в свое время уже получил форму, заполнил её и сейчас от программы требуется сохранить новую запись. Для этого нужно знать метод, с помощью которого отправлен запрос, и если это метод POST, то необходимо проверить полученные данные (если не заполнено одно из полей, вернуть обратно страницу с формой и уведомить об ошибке) и, если проверка пройдена, создать в базе данных новую запись.

if request.method == "POST":

# обработать данные формы, если метод POST

form = {

'text': request.POST["text"], 'title': request.POST["title"]

}

# в словаре form будет храниться информация, введенная пользователем

if form["text"] and form["title"]:

# если поля заполнены без ошибок

Article.objects.create(text=form["text"], title=form["title"], author=request.user)

return redirect('get\_article', article\_id=article.id)

# перейти на страницу поста

else:

# если введенные данные некорректны

form['errors'] = u"Не все поля заполнены"

return render(request, 'create\_post.html', {'form': form})

else:

# просто вернуть страницу с формой, если метод GET

return render(request, 'create\_post.html', {})

Для того, чтобы перейти на главную страницу, в Django из представления можно вернуть результат функции redirect(), которой в качестве параметра достаточно передать имя url (имя для каждого url определяется в файле urls.py в параметре name) и указать, как заполнить именованные группы в регулярном выражении. Теперь перейдем к реализации шаблона с формой:

<!DOCTYPE html>

<html>

<head>

<title>Django Public Blog - создать статью</title>

</head>

<body>

<div class="content">

<h1>Написать статью</h1>

<form method="POST">{% csrf\_token %}

<input type="text" name="title" placeholder="Название статьи" value="{{ form.title }}">

<textarea name="text" placeholder="Текст статьи">{{ form.text }}</textarea>

<input type="submit" value="Сохранить">

</form>

{{ form.errors }}

</div>

</body>

</html>

Тег <form> используется для объединения нескольких полей ввода, чтобы при отправлении, в запрос добавить все данные, которые ввел пользователь. Это особенно важно, когда на одной странице присутствуют сразу несколько полей ввода, которые нужно отправлять независимо (например, если на странице сразу присутствуют и поля авторизации (поля логина и пароля) и поля, например, добавления комментария). Если пользователь оставляет комментарий, на сервер не нужно отправлять данные об авторизации и, наоборот, если пользователь входит в свою учетную запись, не нужно отправлять содержимое поля «комментарий».

Внутри тега <form> находится специальный шаблонный тег {% csrf\_token %}, который добавляет специальное скрытое поле в форму, это поле легко увидеть, если посмотреть содержимое формы через консоль отладки. Затем следуют два главных элемента: <input> (для ввода названия статьи, всегда имеет однострочную форму) и <textarea> (для ввода самой записи, тег позволяет вводить текст на нескольких строках). Значение по умолчанию для тега <input> необходимо указывать в атрибуте value, а для тега <textarea> достаточно написать любой текст между открывающим тегом и закрывающим. Также в форму добавлена кнопка для отправки данных на сервер, а после перечисления всех полей указываются все допущенные пользователем ошибки (если эти ошибки имели место быть).

**Задание:**

* Создайте несколько статей через созданную форму;
* Создайте стили, подключив CSS-файл к шаблону;
* Добавьте проверку на то, что введенное название статьи уникально;
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).

# **Лабораторная работа №6**

Django предоставляет систему аутентификации и авторизации пользователя, реализованную на основе фреймворка работы с сессиями. Система аутентификации и авторизации позволяет вам проверять полномочия пользователей и определять какие кому соответствуют. Код данного приложения находится в пакете django.contrib.auth.

В соответствии с идеологией Django система аутентификации является очень общей и не предоставляет некоторые возможности, которые присутствуют в других системах веб-аутентификации. Решениями некоторых общих задач занимаются пакеты сторонних разработчиков, например, защита от подбора пароля (через стороннюю библиотеку OAuth).

*Создание формы, шаблона и представления для регистрации*

Модель пользователя находится в модуле django.contrib.auth.models.User, можете ознакомиться с исходным кодом модели. Для регистрации пользователя достаточно воспользоваться методом create\_user (метод можно посмотреть в объявлении модели User в исходном коде фреймворка). Пример использования функции:

from django.contrib.auth.models import User

User.objects.create\_user("user1", "user@mail.ru", "user\_password")

В функцию передается 3 параметра: логин, email и пароль. Для того, чтобы пользователь смог отправить эти данные, необходимо создать страницу с формой регистрации и обработать вводимые значения. Не забудьте выполнять проверку на предмет того, имеется ли уже в базе данных пользователь с таким логином. Чтобы это проверить, достаточно попробовать найти его в базе данных с помощью метода get, который вызовет исключение, если объекта не существует:

try:

User.objects.get(username=username)

# если пользователь существует, то ошибки не произойдет и программа # удачно доберется до следующей строчки

print "Пользователь с таким именем уже есть"

except User.DoesNotExist:

print "Этот логин свободен"

**Задание:**

* Создайте шаблон и настройте адрес для отображения формы регистрации;
* Создайте представление, которое будет обрабатывать поступающие запросы и регистрировать новых пользователей. Не забудьте сделать проверку на то, что отправленные поля не являются пустыми, а введенное имя пользователя уникально;
* Создайте стили, подключив CSS-файл к шаблону;
* Добавьте в шапку страницы всех записей и страницы для определенных статей, ссылку на регистрацию в верхнем правом углу (стиль ссылки сделать такой же, как у ссылки “Все статьи” на собственных страницах постов в предыдущих работах).

*Создание формы, шаблона и представления для авторизации*

Для того, чтобы пользователь мог войти в систему, он должен пройти процесс аутентификации (проверка подлинности предъявленного пользователем идентификатора, т.е. имеется ли в базе данных такая пара логин – пароль) и авторизации (предоставление определённому лицу прав на выполнение определённых действий, процесс проверки (подтверждения) данных прав при попытке выполнения этих действий). Для этого можно воспользоваться функцией authenticate из модуля django.contrib.auth. Данная функция принимает два параметра: логин и пароль и сверяет их с базой данных, если такая пара логин – пароль существует, то метод возвращает объект User, иначе возвратит None:

from django.contrib.auth import authenticate

user = authenticate(username="user1", password="user\_pass")

После удачной аутентификации, можно авторизовать пользователя функцией login из модуля django.contrib.auth. Функция принимает два параметра: текущий объект запроса и объект пользователя):

from django.contrib.auth import login

login(request, user)

Иначе, если аутентификация оказалось неудачной и вместо объекта User вернулся None, то пользователю нужно снова вернуть форму входа в систему, при этом вывив сообщение о том, что такого аккаунта не существует.

**Задание:**

* Создайте шаблон и настройте адрес для отображения формы авторизации;
* Создайте представление, которое будет обрабатывать поступающие запросы и авторизовывать зарегистрированных пользователей. Не забудьте сделать проверку на то, что отправленные поля не являются пустыми, а введенные имя пользователя и пароль соответствуют одному из зарегистрированных аккаунтов;
* Создайте стили, подключив CSS-файл к шаблону;
* Загрузите ваш проект на любой гит-репозиторий (GitHub, GitLab, Google Code, Bitbucket и т.п.).

# **Лабораторная работа №7**

*Изучение основ JavaScript, создание простейших функций и использование базовых операторов*

JavaScript – язык программирования с динамической типизацией, очень широко распространен в веб-разработке. JavaScript изначально создавался для того, чтобы сделать web-странички «живыми». Программы на этом языке называются скриптами. В браузере они подключаются напрямую к HTML и, как только загружается страница – тут же выполняются. Сценарии на языке JavaScript загружаются браузером с сервера и выполняются на компьютере пользователя, соответственно для многих задач при работе на JS не нужно отсылать лишних запросов на сервер и ждать ответа для обработки. В JS есть широкие возможности обмена информацией с сервером без перезагрузки веб-страницы, что активно используется во многих современных приложениях.

Для просмотра JavaScript-кода веб-страницы, откройте в браузере панель разработчика, перейдите во вкладку Source и затем выберете файл, код которого вам нужен.

Приступим к написанию первого скрипта на JavaScript. Скопируйте ваш проект из прошлой лабораторной работы в директорию **lab7**. Затем, в папке **articles/static/** создайте папку **js**. В директории **js** создайте файл helloworld.js.

Для того, чтобы подключить JavaScript-файл к веб-странице, необходимо добавить тег в html-шаблон:

<script src="{{ STATIC\_URL }}js/helloworld.js"></script>

Данный тег может находиться в любом месте файла (но не за пределами <head> и <body>), однако, желательно скрипты подключать ближе к закрывающемуся тегу <body>, так как таким образом загрузка сценариев не будет мешать загрузке остального контента (текст, картинки, стили, шрифты).

Затем подключите файл helloworld.js к html-шаблону **archive.html**. Запустите проект и пройдите по адресу http://127.0.0.1:8000/. Запустите панель отладчика, перейдите на вкладку Network и убедитесь, что js-файл есть в списке загруженных файлов, как это показано на рисунке 7.

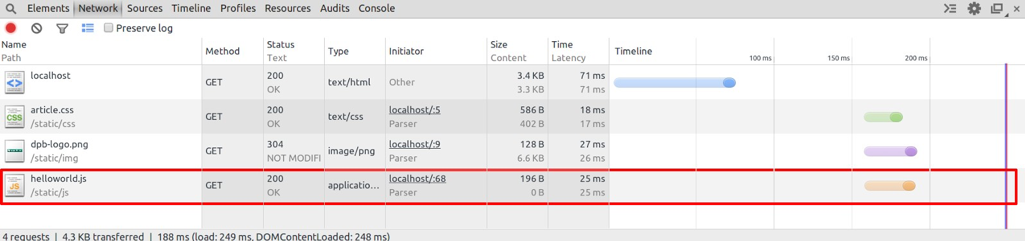


Рисунок 7. JS-файл в списке загруженных

После того, как файл был загружен, напишите его исходный код. Выполните задание, схожее с заданием из лабораторной работы №1. Создайте список студентов с полями: имя, фамилия, группа, оценки.

*Пример*:

var groupmates = [

{

"name": "Александр",

"surname": "Иванов",

"group": "БВТ1702",

"marks": [4, 3, 5]

},

{

"name": "Иван",

"surname": "Петров",

"group": "БСТ1702",

"marks": [4, 4, 4]

},

{

"name": "Кирилл",

"surname": "Смирнов",

"group": "БАП1801",

"marks": [5, 5, 5]

}

];

Для вывода переменной на экран воспользуйтесь методом console.log(), в качестве аргументов функция принимает то, что нужно вывести на экран (в консоль). Например: console.log(groupmates);

Перезагрузите страницу в браузере и в панели разработчика перейдите во вкладку Console. Там можно увидеть созданный вами массив, но в свернутом виде. Для того, чтобы посмотреть содержимое массива, нажмите на стрелку перед массивом.



Рисунок 8. Созданный массив в свернутом виде

Такая форма вывода не очень удобна. Создайте функцию, которая будет выводить содержимое вашего массива в виде таблицы, наподобие таблицы в лабораторной работе №1.

var rpad = function(str, length) {

// js не поддерживает добавление нужного количества символов

// справа от строки, т.е. аналога ljust из Python здесь нет

str = str.toString(); // преобразование в строку

while (str.length < length)

str = str + ' '; // добавление пробела в конец строки return str; // когда все пробелы добавлены, возвратить строку

};

var printStudents = function(students){

console.log(

rpad("Имя", 15),

rpad("Фамилия", 15),

rpad("Группа", 8),

rpad("Оценки", 20)

);

// был выведен заголовок таблицы

for (var i = 0; i<=students.length-1; i++){

// в цикле выводится каждый экземпляр студента

console.log(

rpad(students[i]['name'], 15),

rpad(students[i][surname], 15),

rpad(students[i]['group'], 8),

rpad(students[i]['marks'], 20)

);

}

console.log('\n'); // добавляется пустая строка в конце вывода

};

printStudents(groupmates);

В данном случае, функция rpad была реализована для обеспечения форматирования строки с помощью добавления пробелов, так как в JS нет встроенных средств форматирования. Функция printStudents – выводит список всех студентов.

**Задание:**

* Напишите функцию, которая фильтрует студентов по группе. Наименование группы, по которой будет проводиться фильтрация, вводится пользователем с клавиатуры.;
* Вам необходимо написать функцию фильтрации студентов по средней оценке, так, чтобы на экран выводился список студентов, средний балл которых выше заданного. Средний балл, по которому будет проводиться фильтрация, вводится пользователем с клавиатуры.

*Работа с элементами DOM с помощью JavaScript*

Основным инструментом работы и динамических изменений на странице является DOM (Document Object Model) – объектная модель, используемая для XML/HTML-документов.

Согласно DOM-модели, документ является иерархией, деревом. Каждый HTML-тег образует узел дерева с типом «элемент». Вложенные в него теги становятся дочерними узлами. Для представления текста создаются узлы с типом «текст».

DOM – это представление документа в виде дерева объектов, доступное для изменения через JavaScript.

С помощью JavaScript можно добавить динамичности веб-странице. Добавьте вашей странице со списком статей возможность скрыть и развернуть информацию по каждому посту. Пример отображения представлен на рисунке 9. При нажатии на кнопку «свернуть», должны скрываться имя автора, время создания и сам текст, кнопка «свернуть» меняется на «развернуть».

Для начала добавьте к каждой статье в html-шаблоне кнопку «свернуть», при этом добавив ей класс fold-button. Затем создайте файл **fold-post.js** в папке **articles/static/js/**.



Рисунок 9. Страница отображения всех статей с возможностью сворачивания и разворачивания информации

Затем создайте функцию, которая будет вызываться каждый раз, когда пользователь будет кликать по кнопке, функция должна выводить сообщение в консоль. JS-скрипт:

var foldBtns = document.getElementsByClassName("fold-button");

for (var i = 0; i<foldBtns.length; i++){ foldBtns[i].addEventListener("click", function(event) {

console.log("you clicked ", event.target);

});

}

document – глобальная переменная, для нее вызывается метод getElementsByClassName(), который ищет все элементы с указанным классом в структуре DOM и возвращает их массив. Затем цикл for проходит по всем элементам массива и к каждому элементу, с помощью метода addEventListener, добавляет обработчик событий click, который принимает два параметра: название события и функцию, которая будет вызываться при каждом событии. Обратите внимание, что функция-обработчик может принимать аргумент (имя которого не имеет значения), и именно в этот аргумент будет вноситься объект, содержащий основную информацию о событии (ссылку на сам элемент, по которому кликнули, координаты курсора в момент события, какие были зажаты в этот момент клавиши и т. п.).

Зайдите в панель разработчика в вашем браузере и поставьте брейкпоинт (точку останова, breakpoint) на пятой строке, нажав на номер строки (строка, в которой указан код: console.log("you clicked ", event.target);). Затем нажмите на кнопку «свернуть» для того, чтобы запустить код на исполнение. Теперь вам доступна отладка скрипта пошагово.

Далее приступим к разработке и реализации появления и скрывания информации о статье. Добавьте данный код во внутрь функции-обработчика:

foldBtns[i].addEventListener("click", function(e) {

event.target

.parentElement

.getElementsByClassName('article-author')[0]

.style.display = "none";

event.target

.parentElement

.getElementsByClassName('article-created-date')[0]

.style.display = "none";

event.target

.parentElement

.getElementsByClassName('article-text')[0]

.style.display = "none";

e.target.innerHTML = "развернуть";

});

Сначала в операторе с помощью атрибута parentElement определяется родитель элемента по которому произошел клик, затем внутри этого родителя производится поиск всех элементов с классом “article-author” с помощью метода getElementsByClassName(). Этот метод возвращает массив элементов, потому что по умолчанию один и тот же класс может иметь несколько элементов (даже если в узле DOM-а, где производится поиск, есть всего один элемент с указанным классом, метод все равно вернет список, хоть и с одним элементом). В соответствии с созданной вами html-разметкой внутри родителя one-post только у одного блока есть такой класс, значит метод возвращает список с одним элементом, доступ к которому можно получить по нулевому номеру. Теперь остается этому элементу с помощью атрибутов style и display указать значение none. Именно css-свойство display: none позволяет скрывать элемент на странице.

После необходимо у элемента, по которому кликнули, изменить текст, эта операция выполняется с помощью атрибута innerHTML.

После реализации скрытия необходимо вернуть исходное состояние при повторном клике на эту кнопку. Так как при любой ситуации функция-обработчик, созданная вами ранее, будет вызвана, нужно именно в ней решить эту проблему. Сделайте так, чтобы при нажатии на кнопку «свернуть», самой кнопке добавлялся класс folded: e.target.className = "fold-button folded";

Обратите внимание, что у кнопки уже есть класс fold-button, который не следует удалять, поэтому через пробел был добавлен описывающий текущее состояние класс. Однако, раз у кнопки, которая уже была нажата, есть дополнительный класс, значит вы можете в начале функции-обработчика проверять наличие этого класса, и если он есть, то не скрывать информацию о записи, а наоборот показывать её. Для этого добавьте проверку и в случае, если у кнопки есть класс folded, сделайте вновь видимой скрытую информацию:

foldBtns[i].addEventListener("click", function(e) {

if (e.target.className == "fold-button folded"){

e.target.innerHTML = "свернуть";

e.target.className = "fold-button";

var displayState = "block";

}

else{

e.target.innerHTML = "развернуть";

e.target.className = "fold-button folded";

var displayState = "none";

}

event.target

.parentElement

.getElementsByClassName('article-author')[0]

.style.display = displayState;

event.target

.parentElement

.getElementsByClassName('article-created-date')[0]

.style.display = displayState;

event.target

.parentElement

.getElementsByClassName('article-text')[0]

.style.display = displayState;

});

**Задание:**

* Сделайте другую реализацию функции-обработчика, которая была бы более компактна. Предложенный вариант довольно громоздок, однако он хорошо иллюстрирует некоторые возможности манипулирования DOM с помощью JavaScript. Весь описанный выше функционал можно реализовать с помощью css-классов и их изменения через функцию-обработчик. Сделайте так, чтобы класс folded добавлялся не кнопке, по которой кликнули, а родителю всего поста (это тот самый элемент, у которого в html-разметке установлен класс one-post). В таком случае свойство display можно изменять через css-стили:

.one-post.folded .article-author{

/\* данный стиль применится только для элементов класса

.article-author, у которых родитель с классом one-post имеет также класс folded \*/

display: none;

}

* По аналогии с разработанной вами функцией, установите через CSS-стили «исчезновение» остальных элементов поста.