Инструментальные средства информационных систем

Группа: БИН-19-1

Автор: Сафиуллина Н.Ф.

Лабораторная работа 7 Hello Web

Цель: Познакомится с принципами создания web-приложений.

Задачи:

1. Установка и настройка сервера Apache.
2. Создание web-приложение с помощью Django.
3. Создание Telegram bot.

Примечание: воспользуйтесь материалом лекции № 6 Инструментальные средства web-разработки.

Содержание:

1 Apache

Apache – это бесплатное открытое программное обеспечение для создания web-сервера.

Задание 1

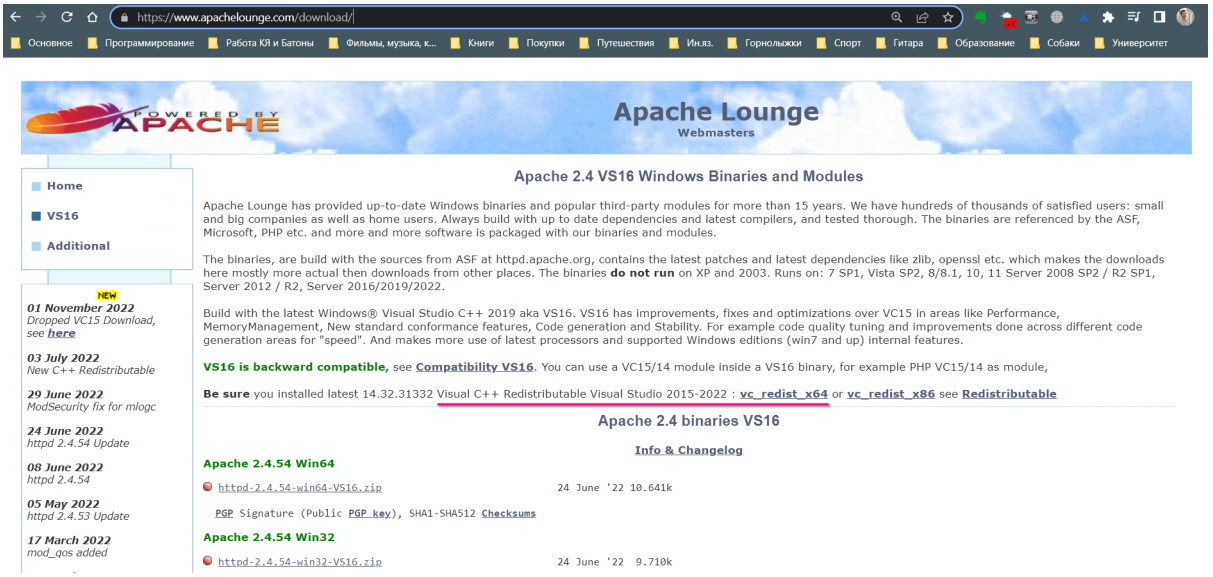
Установите и настройте сервер Apache, сделайте так, чтобы стартовая страница при обращении к сайту по адресу http://127.0.0.1/ генерировалась с помощью скрипта Python. Сделайте наполнение скрипта любым каким вам хочется.

1.1 Установка Apache24

Официальный сайт Apache предоставляет только исходный код своего HTTP-сервера. Но там же сказано, что если вы не можете скомпилировать HTTP-сервер Apache самостоятельно, вы можете получить бинарный пакет из многочисленных бинарных дистрибутивов, доступных в Интернете. Мы воспользуемся Apache Lounge:

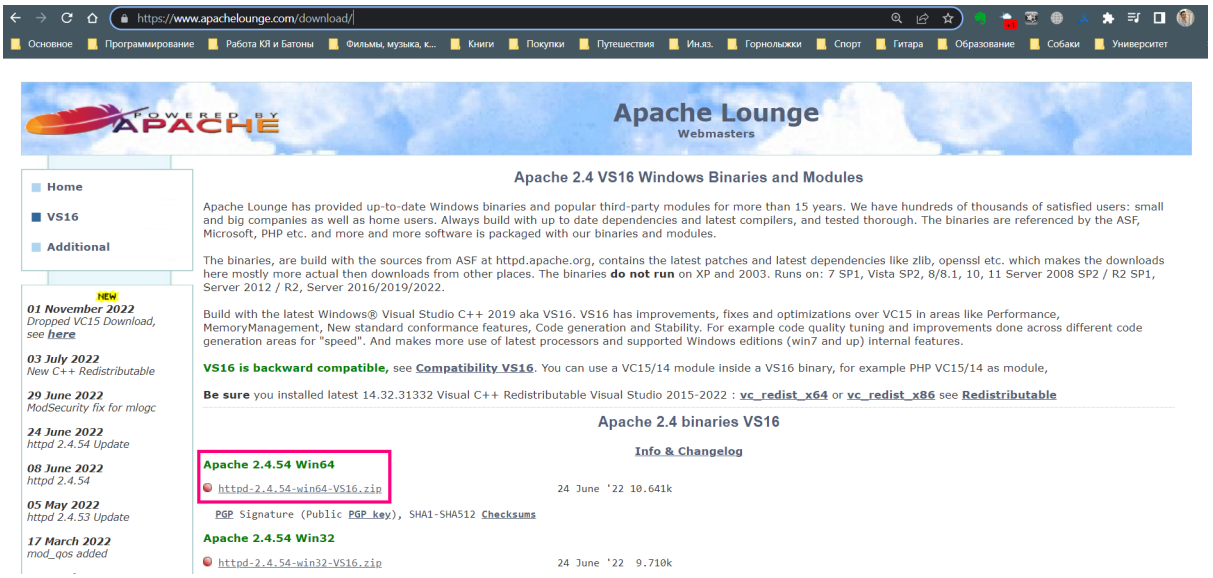
https://www.apachelounge.com/download/

Сначала нужно установить последнюю версию Visual C++ Redistributable for Visual Studio 2015-2019:



Если у вас 64 битная операционная система Windows, то скачать нужно `vc_redist.x64`, а если у вас 32 битная операционная система – `vc_redist_x86` .

Далее скачиваем архив содержащий Apache:



Получим архив:

`httpd-2.4.54-win64-VS16.zip`

распакуем его в корень диска `C:\`.

Можно выбрать и другое место, но тогда надо будет поменять путь в настройках Apache в файле httpd.conf параметр Define SRVROOT должен быть равен местоположению распакованного Apache24. Файл конфигурации находится по относительному пути:

```
C:\Apache24\conf\httpd.conf
```

Настройку сервера мы рассмотрим чуть ниже. А сейчас запустим и проверим наш север и проверим его работу. В командной строке переходим в папку Apache24\bin:

```
C:\>cd C:\Apache24\bin
```

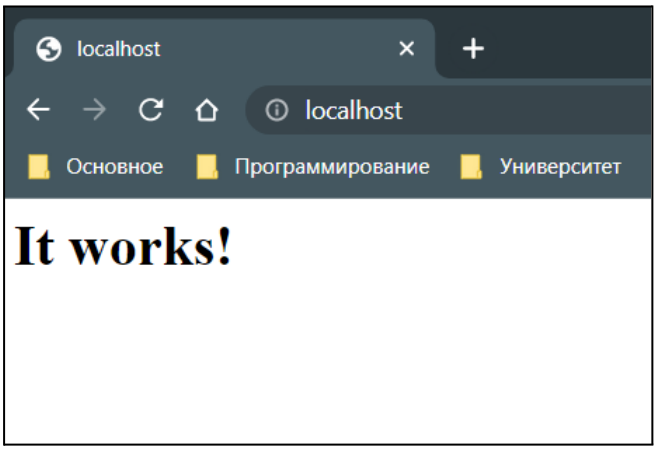
Запустим сервер:

```
C:\Apache24\bin>httpd.exe
```

Если курсор переместился на новую строку и строка осталась пустой, значит наш сервер запущен. Проверим его работу, откройте браузер и введите адрес:

http://localhost

Мы должны увидеть страницу приветствия:

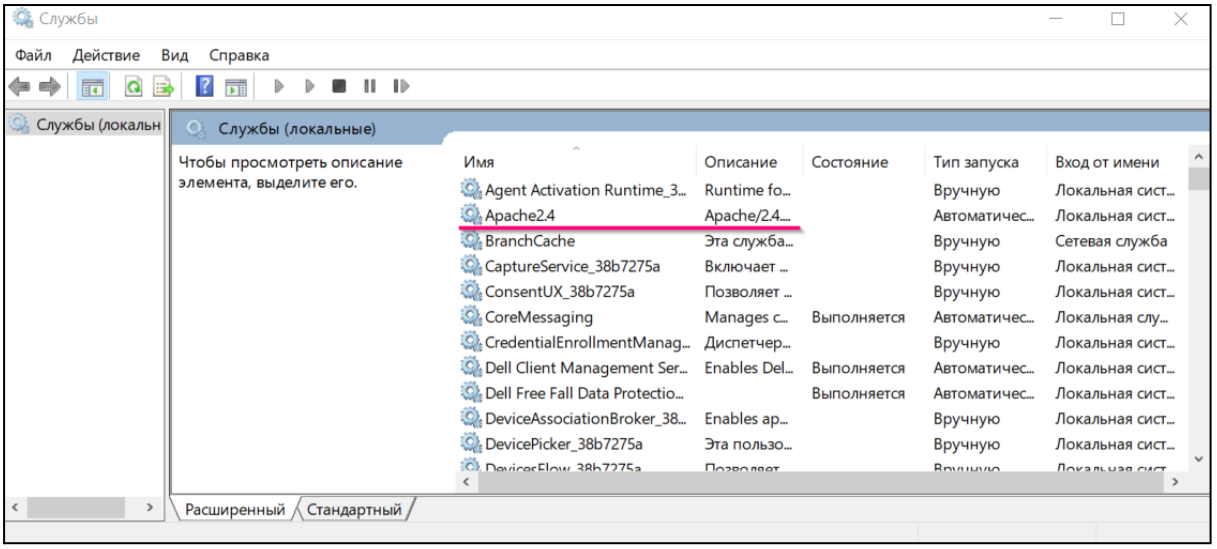


Чтобы остановить сервер Apache в командной строке нужно нажать Ctrl+C. Остановка сервера занимает несколько секунд.

Следующее делать не обязательно, это просто для информации. Можно установить Apache как сервис ОС Windows. Для этого нужно выполнить командной строке запущенной от администратора:

httpd.exe -k install

Тогда, после успешной установки, служба появится в списке:



Как всегда посмотреть все доступные команды с кратким описанием можно в справке:

```
C:\Apache24\bin>httpd -h
```

1.2 Базовая настройка Apache

Файл настроек, или конфигурации, находится:

```
Apache24\conf\httpd.conf
```

В нем есть строки комментариев начинаются с #, а так же есть строки – директивы, которые тоже начинаются с #, это означает, что директивы отключены.

Директивами называются параметры или опции, которые создают инструкции для web-сервера.

Как и было сказано ранее первую директиву, которую нужно настроить это путь к папке, где хранятся конфигурация сервера, ошибки и протоколы:

```
Define SRVROOT "c:/Apache24"
ServerRoot "${SRVROOT}"
```

Первая строка присваивает значение переменной SRVROOT, которое по умолчанию "C:/Apache24". Вторая строка присваивает настройке ServerRoot значение переменной ${SRVROOT}.

Значение в первой строке нужно заменить на то, куда распаковали архив Apache.

Директива Listen позволяет вам привязать Apache к определённому IP адресу и/или порту. Значение по умолчанию:

```
Listen 80
```

Оно означает, прослушивать 80 порт на любом IP адресе (т.е. любом сетевом интерфейсе), доступном в вашей системе. Вы можете указать конкретный IP адрес, который нужно прослушивать и, следовательно, на запросы которого отвечать:

```
Listen 127.0.0.1:80
```

Можно указать несколько портов:

```
Listen 80
Listen 8080
```

Или несколько IP и портов:

```
Listen 127.0.0.1:80
Listen 127.0.0.1:8080
```

Вы можете использовать любые сочетания, главное правило – порт на указанном интерфейсе (IP) не должен быть занят другой программой. Помните порт 3306 у нас занят MySQL, а 6379 – Redis.

Значение по умолчанию вполне подходит для локального веб-сервера – т.е. здесь можно просто ничего не менять.

Далее идет список модулей. Некоторые из них включены, некоторые выключены. Нам нужно включить модуль **mod_rewrite**. Ищите строку:

```
LoadModule rewrite_module modules/mod_rewrite.so
```

Раскомментируйте ее. Этот модуль выполняет множество функций, одна из них, например, это превращение URL в ЧПУ – в человекопонятные URL.

Следующая директива, которая нам нужна:

```
ServerAdmin admin@example.com
```

Это адрес администратора сервера. Замените на свой адрес электронной почты.

Директива:

```
ServerName www.example.com:80
```

Это имя и порт, которые сервер использует для идентификации самого себя. Замените значение на localhost:

```
ServerName localhost
```

Следующий блок директория:

```
<Directory />
    AllowOverride none
    Require all denied
</Directory>
```

Запрещает доступ к файловой системе вашего сервера. Чтобы открыть доступ к папкам с содержимым, необходимо явно указать их в блоках директориев ниже в файле конфигурации.

Следующий блок перезаписывает предыдущий блок для конкретной папки:

```
DocumentRoot "${SRVROOT}/htdocs"
<Directory "${SRVROOT}/htdocs">
       Options Indexes FollowSymLinks ExecCGI
       AllowOverride None
       Require all granted
</Directory>
```

Директива DocumentRoot – это путь, где расположен ваш сайт. Если ваш контент будет находится в другом месте, нужно не забыть поменять адрес в данном блоке.

Как видно тут используется переменная SRVROOT, для относительного задания пути. Если контент сайта будет находится в другом месте, упоминание переменной нужно удалить.

Директива Options может принимать значения “None”, “All” или любую комбинацию из следующих значений: “Indexes”, "Includes”, “FollowSymLinks”, “SymLinksifOwnerMatch”, “ExecCGI”, “MultiViews”. В нашем примере:

– Index – говорит, что если запрос пришел без указания имени файла, который надо открыть, то откроется файл с именем index (в него можно как раз поместить главную страницу).

– FollowSymLinks – разрешение следовать символьным ссылкам.

– ExecCGI – наличие этой опции позволяет запускать CGI скрипты из данного каталога (но не делайте так в реальном продукте в целях безопасности), cgi скрипты – это скрипт, который работает через интерфейс CGI (common gateway interface), по сути используют стандартную консоль ввода-вывода. Нашим CGI-скриптом будет скрипт на Python.

AllowOverride контролирует какие директивы могут быть указаны в .htaccess (это тоже конфигурационный файл сервера, который заменяет параметры указанные в основном конфигурационном файле, при этом мы не меняем основной файл).

Директива Require all granted открывает доступ к файлам контента.

Блок:

```
<IfModule dir_module>
    DirectoryIndex index.html
</IfModule>
```

Говорит Apache какие файлы индексные, т.е. эти файлы открываются по запросу сайта, если запрос пришел без указания конкретного файла. Но нам надо чтобы запускался наш CGI-скрипт на Python, поэтому меняем значение:

```
<IfModule dir_module>
    DirectoryIndex index.py
</IfModule>
```

“Index” тоже можно заменить на имя скрипта.

В блоке:

```
<IfModule mime_module>
       TypesConfig conf/mime.types
       AddType application/x-compress .Z
       AddType application/x-gzip .gz .tgz
       AddHandler cgi-script .cgi .py
</IfModule>
```

Находим директиву AddHandler, раскомментируйте ее и замените в ней “.cgi” на “.py”, т.е. файлы с расширением “py” теперь будут рассматриваться как CGI-скрипт.

Настройка готова. Перезапустите сервер, чтобы применить настройки.

1.3 Создание CGI-скрипта на Python

Далее просто помещаете в каталог, указанный в директиве DocumentRoot, скрипт Python примерно такого вида (“...” нужно заменить на значение из своих настроек, это путь к интерпретатору Python):

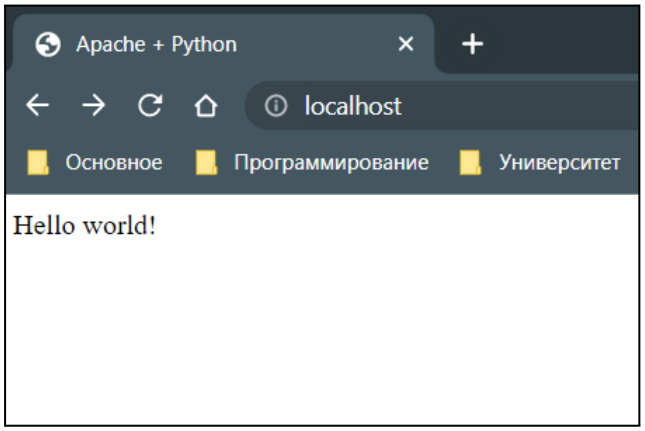
```
#!C:\Users\...\AppData\Local\Programs\Python\Python310\python.exe
print ('Content-type: text/html\n\n')
print ('<html>')
print (' <head>')
print ('   <title>Apache + Python</title>')
print (' </head>')
print (' <body>')
print ('   Hello world!')
```

```
print (' </body>')
print ('</html>')
```

Проверим работу, откройте браузер и введите адрес:

http://localhost

Мы должны увидеть новую страницу:



2 Telegram bot

Задание 2

Создайте Telegram бота. Придумайте ему функциональность и реализуйте ее на Python.

В качестве инструкции воспользуйтесь материалом лекции 6.

3 Web-приложение с помощью Django

Задание 3

Создайте сайт который будет:

– иметь форму ввода, в которой можно будет задать какие-то параметры поиска данных;

– осуществлять поиск данных в нашей базе данных MySQL, развернутой в контейнере my-mysql в Docker, по заданным параметрам.

Используйте следующие компоненты Django:

– встроенный собственный локальный веб-сервер;

– формы;

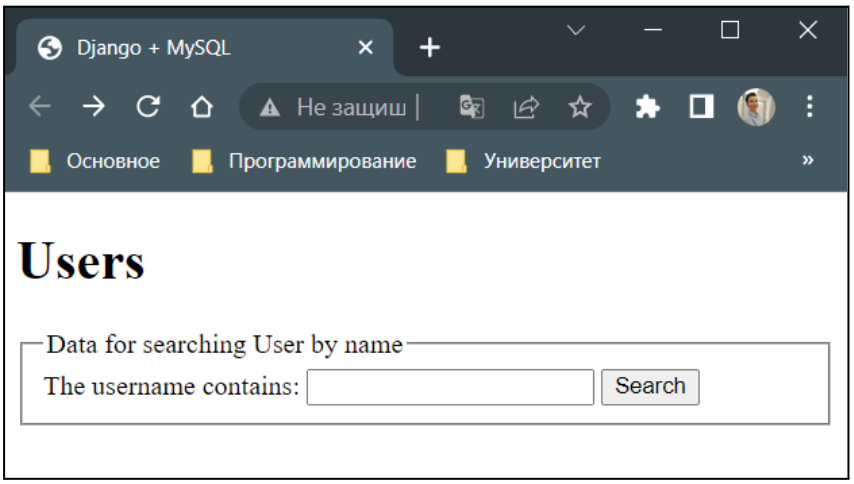
– модели данных;

– шаблоны html.

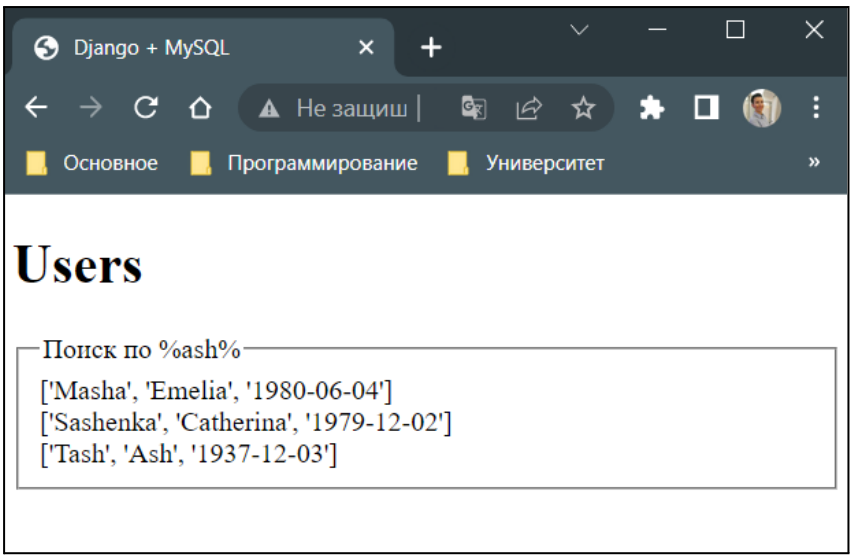
Сделайте что-нибудь отличающиеся от примера, который будет описан ниже. (Например, вы можете добавить выбор метода поиска: начинается с заданной подстроки, содержит её или заканчивается ей; или можно сделать поиск по нескольким полям одновременно.)

Пример. Далее будет описано создание сайта состоящего из двух страниц:

– На первой странице представлена web-форма, для ввода последовательности символов, по которым осуществляется поиск в БД по вхождению этой последовательности символов в поле first_name в таблице users:



– На второй странице осуществляется вывод всех найденных записей. Например, все пользователи в имени которых содержится “ash”. Выводятся first_name, last_name и date_of_birth.



Примечание. Для более подробного ознакомления с работой компонентов Django можно обратиться к следующим источникам:

1. Понятное руководство: https://tutorial.djangogirls.org/ru/
2. Все что нужно знать о Django: https://django.fun/ru/docs/django/4.1/
3. Официальная документация: https://docs.djangoproject.com/en/4.1/

3.1 Установка фреймворка Django

В командной строке выполните команду:

```
C:\>python -m pip install Django
```

Если установка прошла успешно, то команда проверки версии должна выдать нам версию:

```
C:\>python -m django --version
```

Позже нам потребуется установка пакета django в PyCharm.

3.2 Создание виртуальной среды

Прежде чем начать создавать новый проект web-приложения нужно создать для него виртуальную среду. Виртуальную среду можно и не создавать, но это полезно, тогда будет возможность изменять один сайт и не затрагивать

изменения другого сайта. Виртуальную среду можно создать из командной строки (но не торопитесь делать):

```
D:\Python\WebProjects\Project1>python -m venv myvenv
```

myvenv – это имя виртуальной среды, оно может быть любым на ваше усмотрение. Затем в этой виртуальной среде можно создать проект Django:

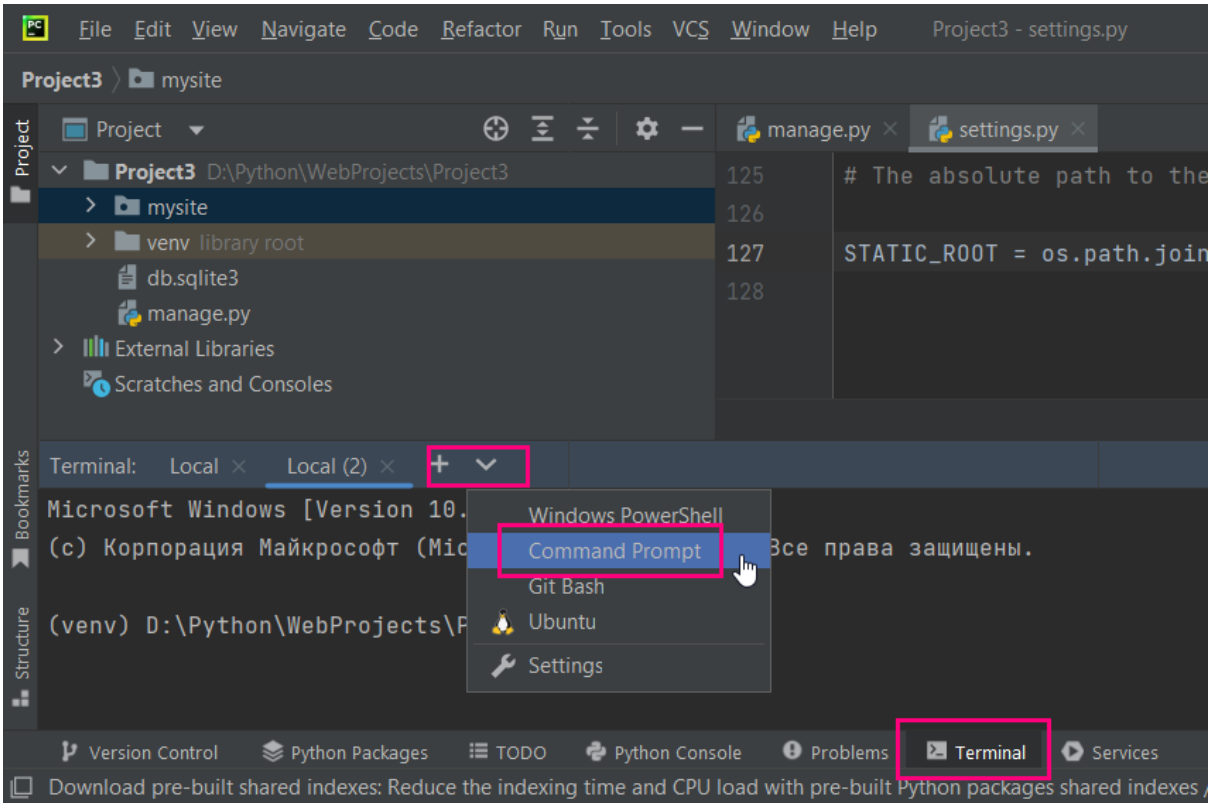
```
D:\Python\WebProjects\Project1>django-admin.exe startproject
mysite .
```

Проще сделать это через PyCharm. Создайте в PyCharm новый проект (опция NewProject). Укажите ему путь, например, “D:\Python\WebProjects\Project3” и все остальные параметры, как у обычного проекта. Только уберите галочку напротив “Create a main.py welcome script”, нам не нужен main.py. В результате будет создана необходимая виртуальная среда.

Если требуется выполнить какую-то команду в командной строке при этом находясь в виртуальной среде, то нам надо запустить виртуальное окружение:

```
D:\Python\WebProjects\Project1>myvenv\Scripts\activate
(myvenv) D:\Python\WebProjects\Project1>
```

Видите (myvenv) в начале строки? Значит виртуальное окружение запущено. Второй способ выполнить команду в виртуальной среде это открыть терминал в PyCharm:



3.3 Создание проекта Django

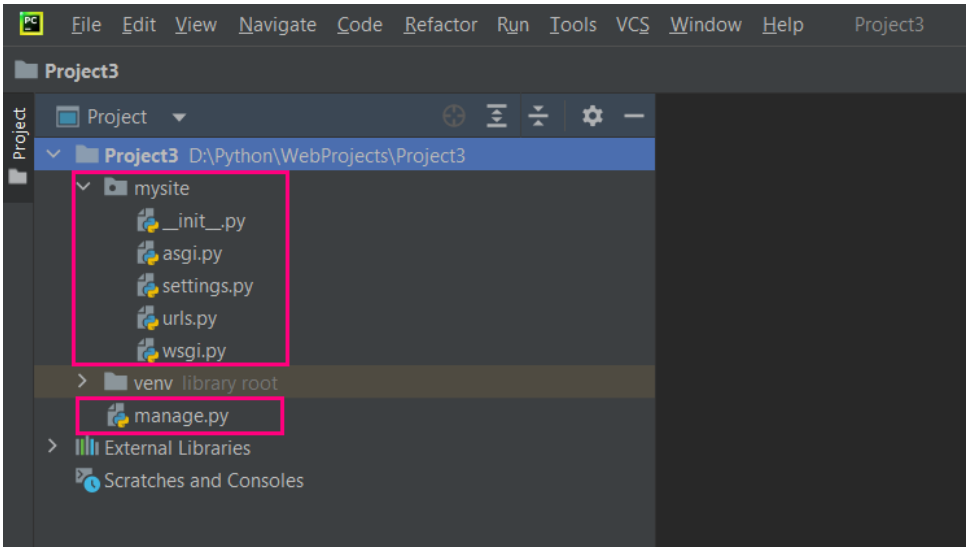
Теперь в командной строке переходим в папку с нашей виртуальной средой, т.е. в папку проекта, созданного в PyCharm:

```
D:\>cd D:\Python\WebProjects\Project3
```

Находясь внутри папки запускаем создание проекта Django, назовем проект mysite:

```
D:\Python\WebProjects\Project3>django-admin.exe startproject mysite .
```

Точка в конце важна, точка говорит, что нужно создать проект в текущей папке. В результате мы увидим, что в PyCharm в дереве нашего проекта Project3 появился новый раздел mysite. Посмотрите какую структуру имеет проект Django, это файлы внутри папки mysite и плюс отдельный файл manage.py, в котором мы найдем содержимое код как в main.

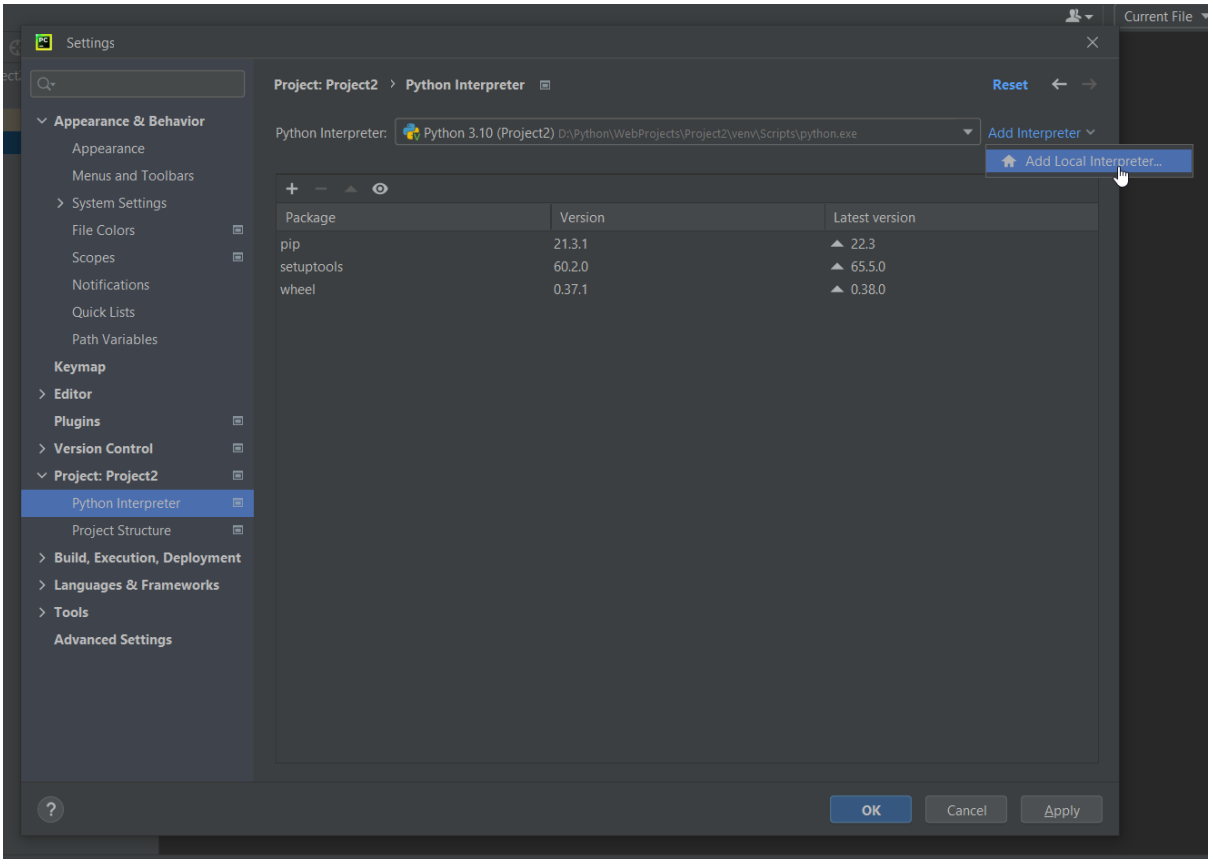


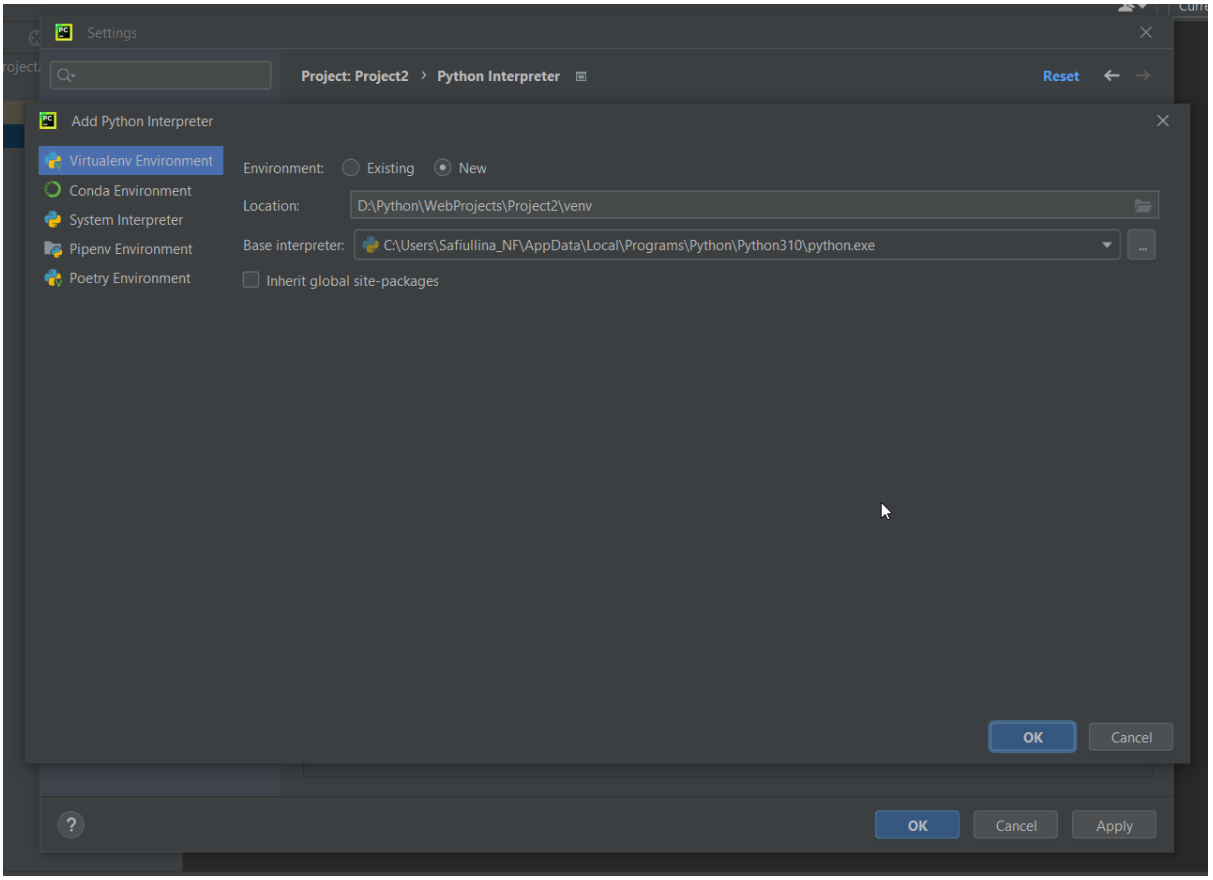
Проект Django создан.

3.3 Настройки окружения и запуск web-сервера

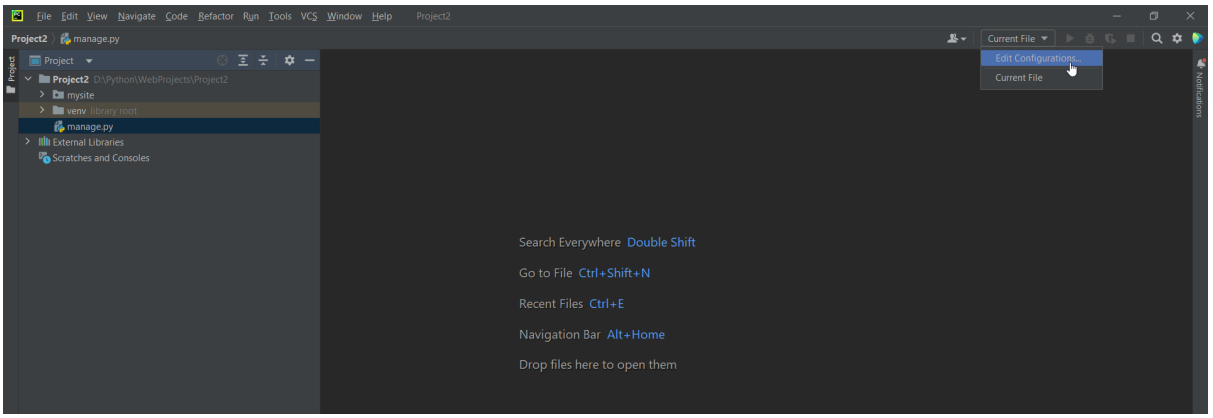
Сделаем настройки окружения необходимые для запуска локального веб-сервера.

Нужно найти Settings, File → Settings, выбираем интерпретатор (если его нет):

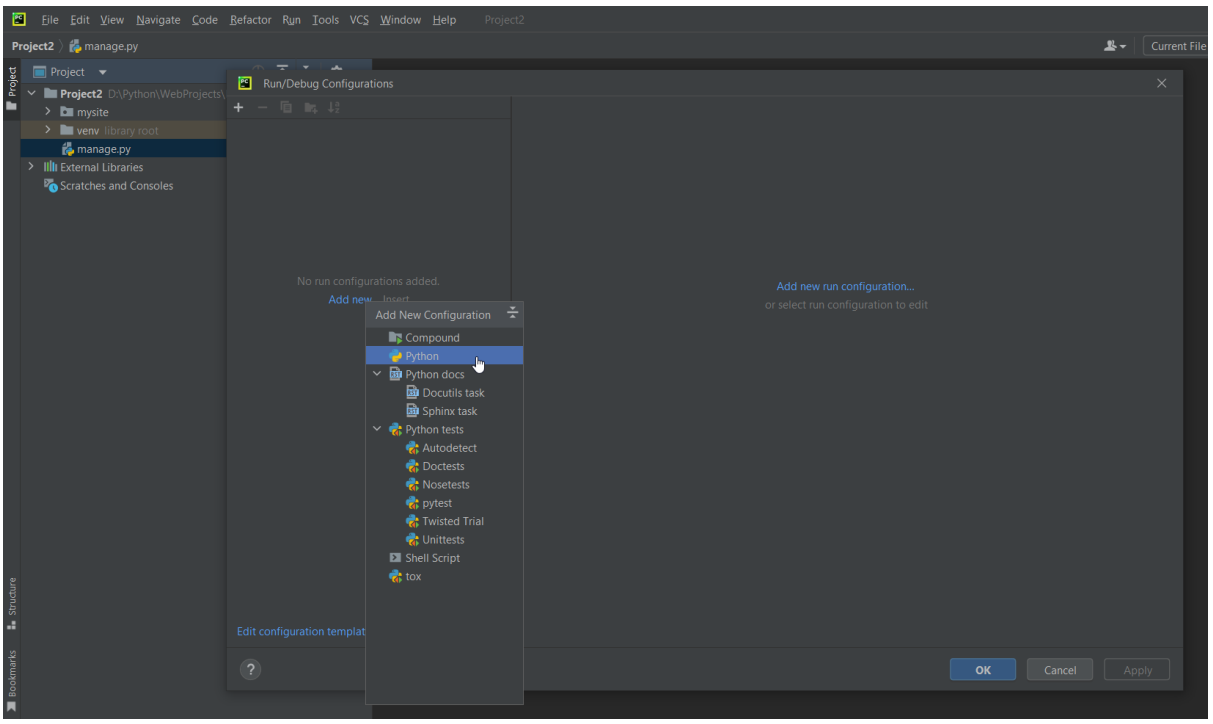

Settings
Current File
Project: Project2 › Python Interpreter
Reset
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Project: Project2
Python Interpreter
Project Structure
Build, Execution, Deployment
Languages & Frameworks
Tools
Advanced Settings
Python Interpreter: Python 3.10 (Project2) D:\Python\WebProjects\Project2\venv\Scripts\python.exe
Add Interpreter
Add Local Interpreter...
Package
Version
Latest version
pip
21.3.1
22.3
setuptools
60.2.0
65.5.0
wheel
0.37.1
0.38.0
OK
Cancel
Apply


Settings
Project: Project2 › Python Interpreter
Reset
Add Python Interpreter
Virtualenv Environment
Conda Environment
System Interpreter
Pipenv Environment
Poetry Environment
Environment: Existing New
Location: D:\Python\WebProjects\Project2\venv
Base interpreter: C:\Users\Safiullina_NF\AppData\Local\Programs\Python\Python310\python.exe
Inherit global site-packages
OK
Cancel
OK
Cancel
Apply

Настраиваем запуск приложения. Приложение будет запускаться файлом manage.py, поэтому стоя на этом файле открываем меню Edit Configurations:



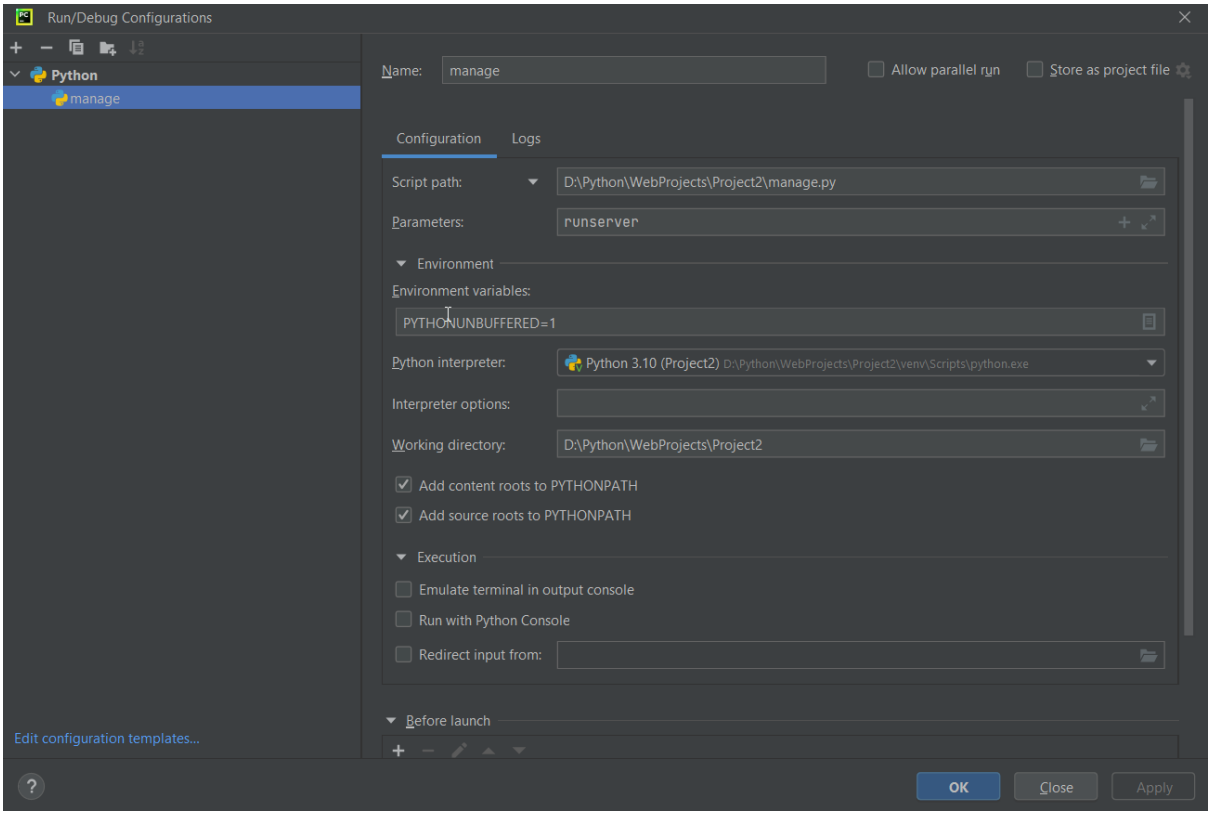
Добавляем новую конфигурацию запуска приложения:



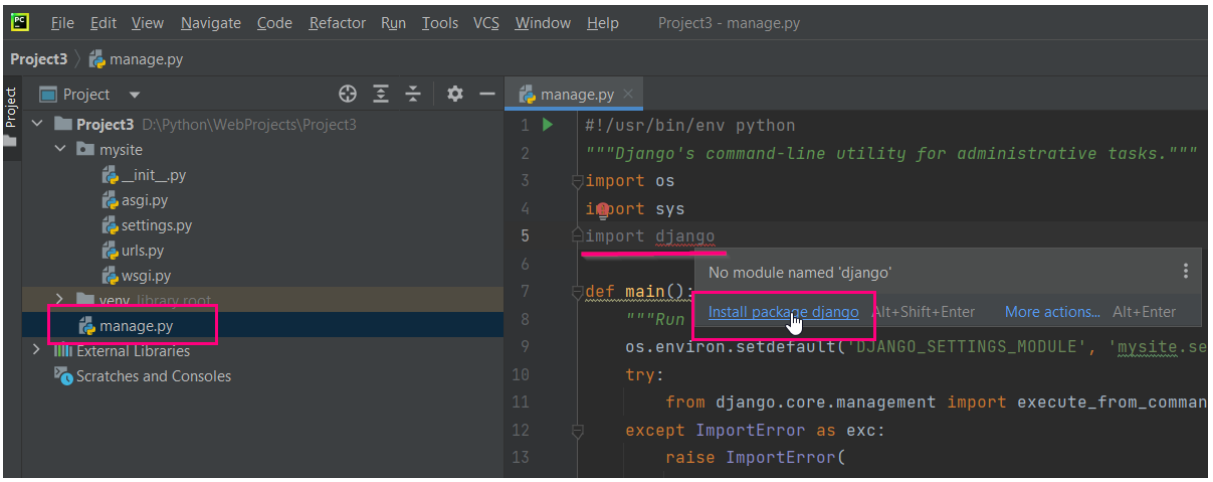
В поле Name укажем manage.

В поле Script path указываем путь до manage.py.

В поле Parameters пишем runserver – это как раз команда запуска web-сервера Django.



Нужно добавить фреймворк Django, т.е. добавляем import django в код:

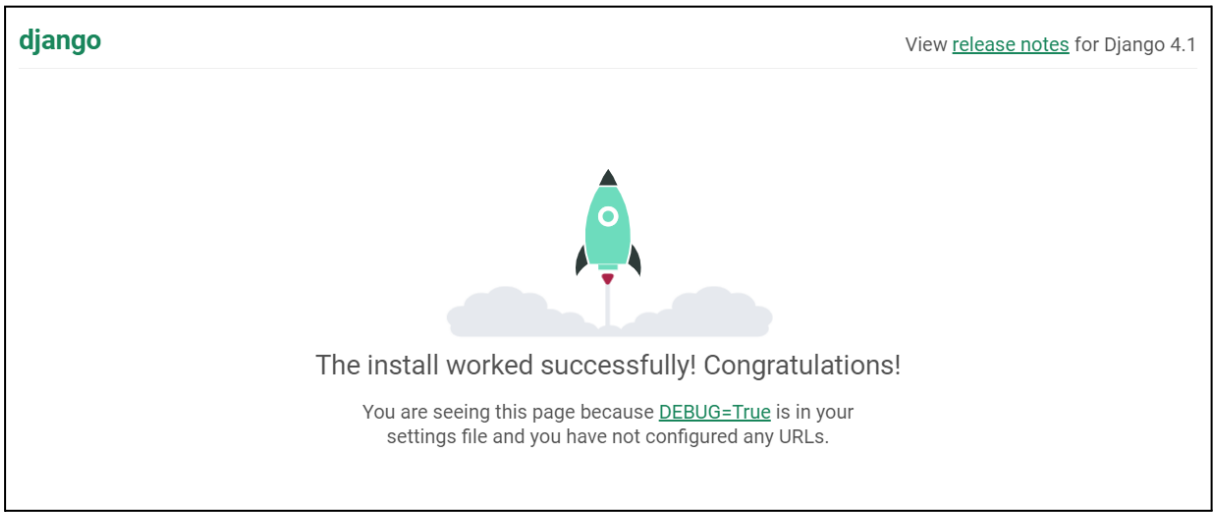


Далее он нам будет тут не нужен, так что можно удалить строку import после импорта.

Всё готово. Стартуем наш веб-сервер, а именно просто запускаем через Run наше приложение. Получим:

```
Performing system checks...

System check identified no issues (0 silenced).

You have 18 unapplied migration(s). Your project may not work
properly until you apply the migrations for app(s): admin, auth,
contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 03, 2022 - 22:51:33
Django version 4.1.3, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

В браузере введите адрес http://127.0.0.1:8000/. Увидите приветственную страницу:



Сервер настроен и запущен. Из командной строки его можно запустить так:

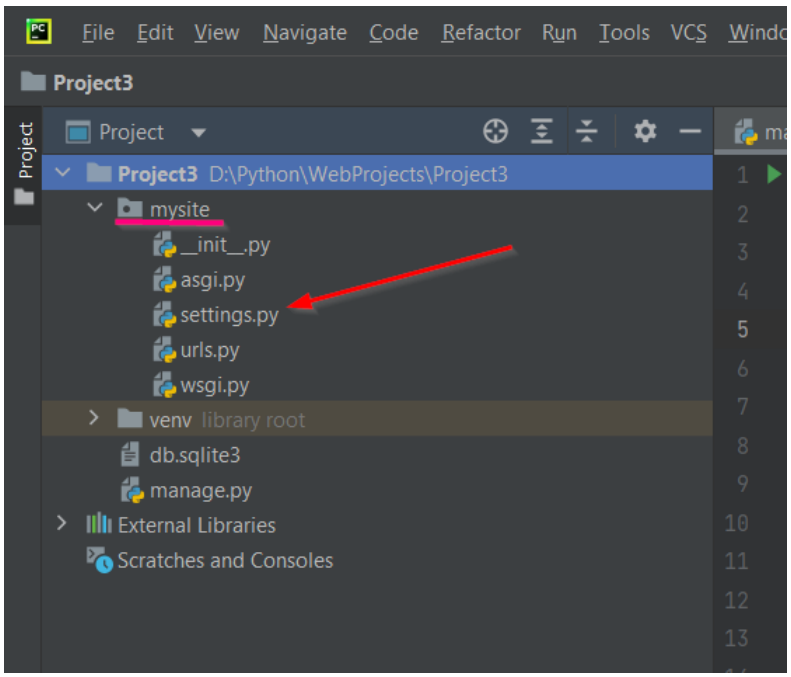
```
D:\Python\WebProjects\Project3>python manage.py runserver 0:8000
```

Остановите веб-сервер кнопкой Stop.

3.4 Настройки web-сайта

Настройки для веб-сайта находятся в скрипте:

mysite/settings.py



Описание возможных настроек можно прочитать тут:

https://docs.djangoproject.com/en/4.1/ref/settings/

Например, мы можем указать язык, чтобы надписи на сайте были на русском языке:

```
LANGUAGE_CODE = 'ru-ru'
```

Можно указать часовую зону.

Давайте зададим путь к папке, где будем хранить статические файлы например, файлы стилей CSS для оформления странички. Добавим в начале файла:

```
import os
```

Это библиотека функций для работы с операционной системой.

Добавим в конце файла переменную:

```
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Таким образом мы зададим путь, где к пути к основному приложению добавляется папка с именем static.

Интересный параметр ALLOWED_HOSTS – где можно задать список IP адресов, имена узлов, доменов, которые обслуживает наш веб-сайт. Пока он нам не нужен, но если нужно сделать сайт доступным извне, значение этого параметра надо будет поменять, например так:

```
ALLOWED_HOSTS = ['127.0.0.1', '192.168.31.159']
```

Настройки веб-сайта сделаны, теперь создадим приложение для нашего сайта.

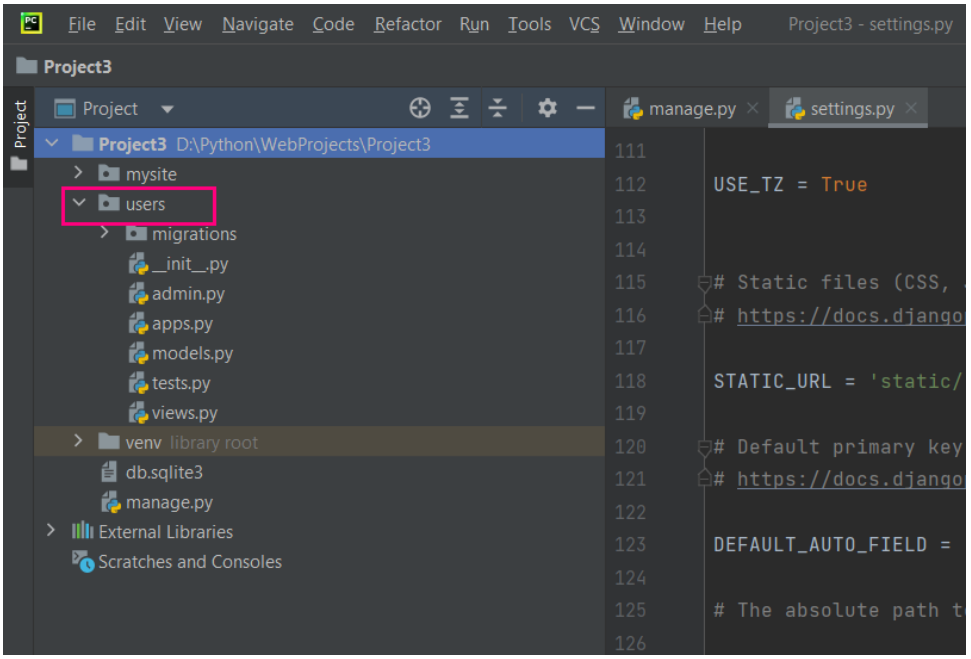
3.5 Создание приложения web-сайта

Приложение веб-сайта будет выполнять нужные нам действия для сайта, будет нести бизнес-логику.

Настал тот момент когда нужно выполнить команду в командной строке, но в виртуальной среде (см.пункт 3.2):

```
(venv) D:\Python\WebProjects\Project3>python manage.py startapp users
```

Если обновить дерево нашего проекта, то мы увидим новый компонент и его структуру – это наше приложение users:



Если приложение создано корректно. Добавим его в список приложений нашего веб-сайта. Список приложений задан в файле настроек:

`mysite/settings.py`

Находим список INSTALLED_APPS и добавляем к списку ‘users’, следующим образом:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'users',
]
```

Приложение в список приложений добавлено, теперь укажем когда его запускать. Приложение должно запускаться, т.е. открывать свою стартовую страничку, когда кто-то обращается на веб-сервер по адресу 'http://127.0.0.1:8000/'. Для этого укажем в файле адресов:

`mysite/urls.py`

Следующие данные:

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

Точнее ‘admin/’ админка уже была создана, добавляем только вторую строку, где импортируем данные об URL-адресах из файла ‘users.urls’, так как мы использовали функцию include, то придется добавить её в импорт в начале файла. Скрипт целиком:

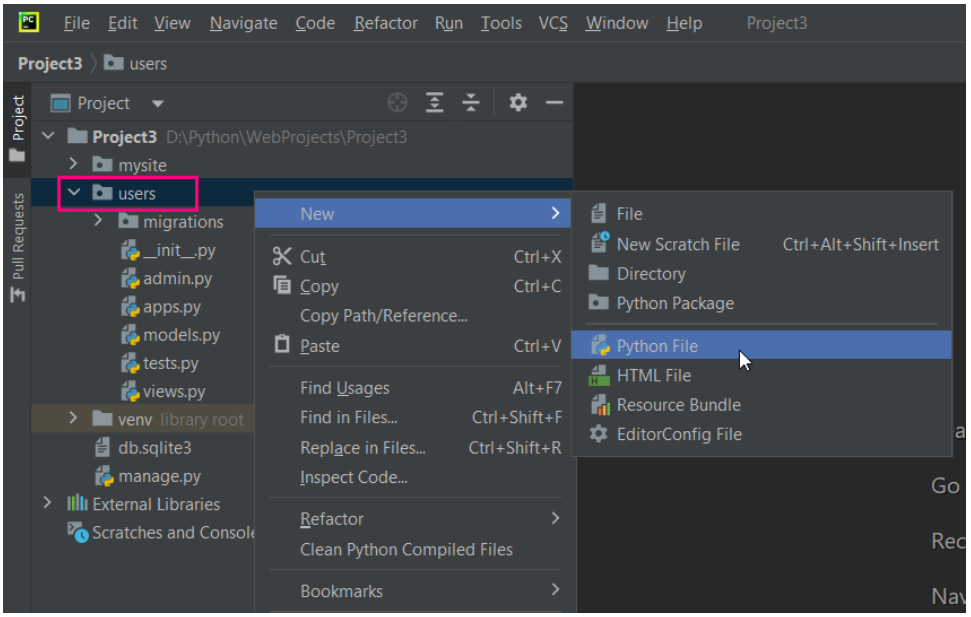
```
from django.contrib import admin
from django.urls import path, include

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

3.6 Наполнение web-приложения

3.6.1 URL-адреса приложения

В приложении users создаем файл urls.py:

`users/urls.py`



Файл urls.py будет пустой, зададим в нем шаблоны адресов и укажем какие обработчики (представления views) запускать:

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.get_name, name='user_name'),
    path('user/search', views.user_by_name, name='user_by_name'),
]
```

Таким образом мы связали конкретный адрес с каким-то конкретным представлением (views), проще говоря с каким-то обработчиком или функцией. Представления содержать бизнес-логику приложения.

У нас две страницы, первая user_name – мы запрашиваем имя пользователя (точнее часть имени), она же стартовая страничка, вторая user_by_name – мы выводим найденные результаты.

3.6.2 Views – описание логики

Откроем файл views.py, там уже есть несколько строк. Добавляем раздел импорт:

```
from django.shortcuts import render
```

```
from django.http import HttpResponseRedirect
from manage import cursor
from .forms import NameForm
from .models import SearchingUsers
```

При чем три последних строки ссылаются на сущности, которых еще нет, это – курсор БД cursor, имя формы ввода NameForm и модель данных SearchingUsers, мы создадим их позже.

Теперь опишем первый обработчик – get_name(), который будет показывать стартовую страничку и будет выводить на ней форму ввода, получать и сохранять данные введенные в эту форму ввода:

```
# создание формы ввода
def get_name(request):
    # если это запрос POST, нам нужно обработать данные формы
    print('Получен метод запроса: ', request.method)
    if request.method == 'POST':
        # создать экземпляр формы и заполнить данным из web-запроса
        form = NameForm(request.POST)
        print('Сырые данные: ', request.POST)

        if form.is_valid():
            print('Введенное значение: ', form.cleaned_data['user_name'])
            # сохраним значение из формы в модель данных django
            SearchingUsers.ByName = form.cleaned_data['user_name']
            # перенаправляем по адресу search
            return HttpResponseRedirect('user/search')
        else:
            print('Form is not valid')

    # Если GET (или любой другой метод), мы создадим пустую форму
    else:
        form = NameForm()

    return render(request, 'users/index.html', {'form': form})
```

Второй обработчик – user_by_name(). Делает следующее:

– формирует запрос к БД и записывает его в строку sql,

– отправляет запрос в БД, подставляя в него данные, которые были введены через форму на первой странице сайта,

– получает ответ от БД, сохраняет его в область cursor,

– записывает результаты в список user_info,

– передаёт список для вывода на второй страничке сайта.

```
# запрос данных из БД
def user_by_name(request):
    print('Получили: ', SearchingUsers.ByName)
    # получим значение из модели и обрамляем в % так как будем искать
вхождение подстроки
    substring = '%' + SearchingUsers.ByName + '%'
    # формирование запроса
    sql = "select first_name, last_name, date_of_birth
           from user where first_name like %s order by first_name"
    # выполнение запроса к БД
    cursor.execute(sql, substring)
    # объявление пустого списка
    user_info = list()

    # формируем список из подсписков,
    # где один подсписок - имя, фамилия, дата рождения.
    for r in cursor:
        #print(r[0], r[1], '\t', r[2])
        #user_info = str(r[0]) + str(r[1]) + str(r[2])
        sublist = [[r[0], r[1], str(r[2])]]
        user_info += sublist

    # открываем страницу с результатами
    return render(request, 'users/results.html',
                  {'user': user_info, 'substring': substring})
```

Render() обозначает “отрисовка”, вспомогательная функция Django, тут мы указали какой шаблон использовать (т.е. какой html файл использовать) и что в нём динамически подставить, т.е. user и substring. Шаблонов HTML мы тоже пока что не имеем.

Теперь заполним пробелы по порядку, создадим: курсор БД, форму ввода, модель данных и шаблоны html.

3.6.3 Соединение с MySQL

Сделаем соединение с БД в самом начале работы сайта, т.е. возвращаемся в файл:

`manage.py`

Импортируем библиотеку для работы с MySQL и добавляем две глобальные переменные, первую для соединения с БД, вторую – для курсора БД (см.лекцию 4 и лабораторную работу 5). Т.е. добавляем следующий код:

```
import pymysql


connect = pymysql.connect(host="127.0.0.1",
                          user="root",
                          password="pass",
                          db="my_db")
cursor = connect.cursor()
```

В основную функцию добавим создание соединения с БД следующим образом:

```
if __name__ == '__main__':
   if connect:
       print("MySQL's connected", end='\n\n')

   main()
```

Если во время запуска веб-сервера соединение с БД пройдет успешно, то мы увидим сообщение об этом.

Кроме того, если теперь посмотреть views.py, то мы увидим, что теперь cursor не подчеркнут красной волнистой линией.

3.6.4 Формы в Django

Создадим новый скрипт в приложении users, точно так же как мы создавали urls.py, теперь создадим:

`users/forms.py`

Файл будет пустой, добавим в него следующие строки:

```
from django import forms
```

```
# объявляем форму с именем NameForm
# принадлежащую классу Form
class NameForm(forms.Form):
   # форма имеет одно поле, строкового типа
   # имеет подпись - label
   # имеет максимальную длину 100 символов
   user_name = forms.CharField(label='The username contains:',
                               max_length=100)
```

Использую формы класса Form и прочих классов форм мы можем описать поля формы и затем воспользоваться ей при отображении нашей странички в html.

3.6.5 Модель

Модель данных – это объект, который позволяет хранить структуры данных, описывает тип полей и их некоторые характеристики. Как правило, модель данных связывают с какой-то конкретной базой данных (мы этого делать не будем).

Описание своей модели помещаем в файл:

`users/models.py`

```
from django.db import models

# создание модели данных класса Model
class SearchingUsers(models.Model):
   # модель имеет одно текстовое поле
   ByName = models.CharField(max_length=100)
```

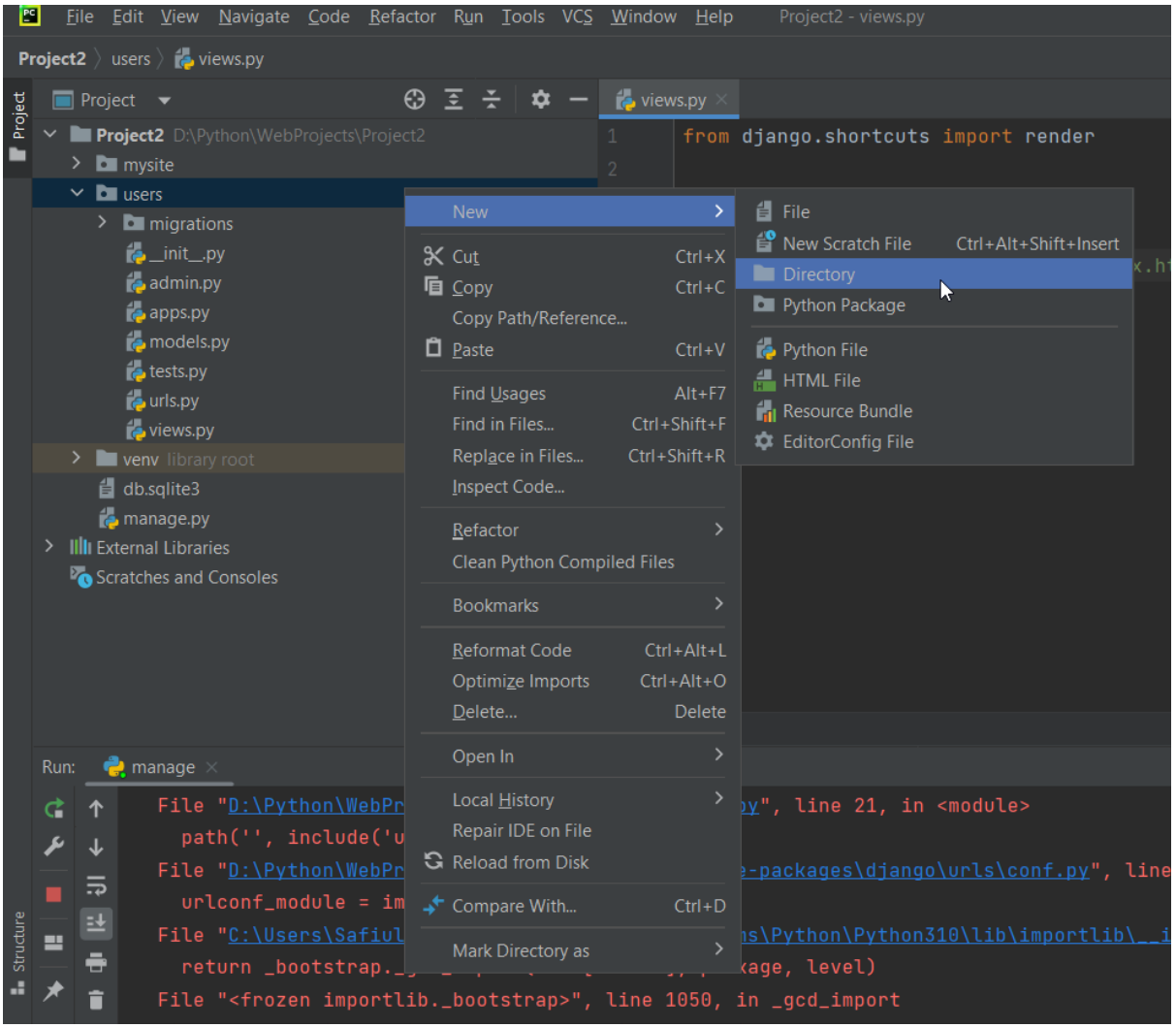
Передача данных между представлениями views в Django возможна через модели данных или базу данных. Именно для этого нам нужна модель. Так как нам нужно передать данные из get_name() в user_by_name().

3.6.6 Шаблоны HTML

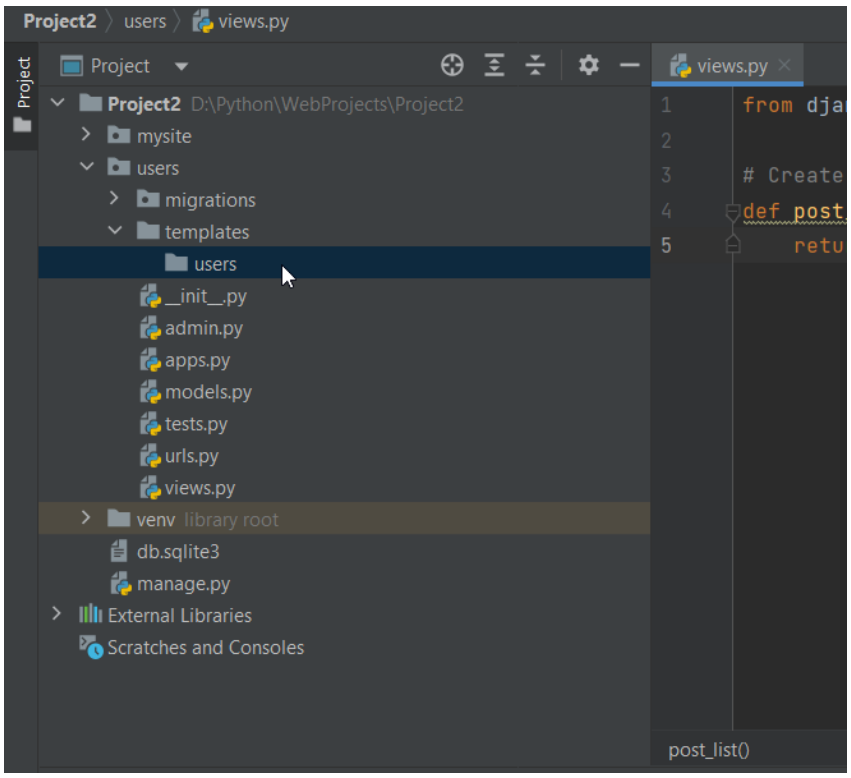
Теперь нарисуем сами странички сайта с помощью html. По общепринятому соглашению шаблоны приложения создаются по следующему пути:

`users/templates/users`

Т.е. нужно в директории users создать директорию templates и в ней создать директорию users.

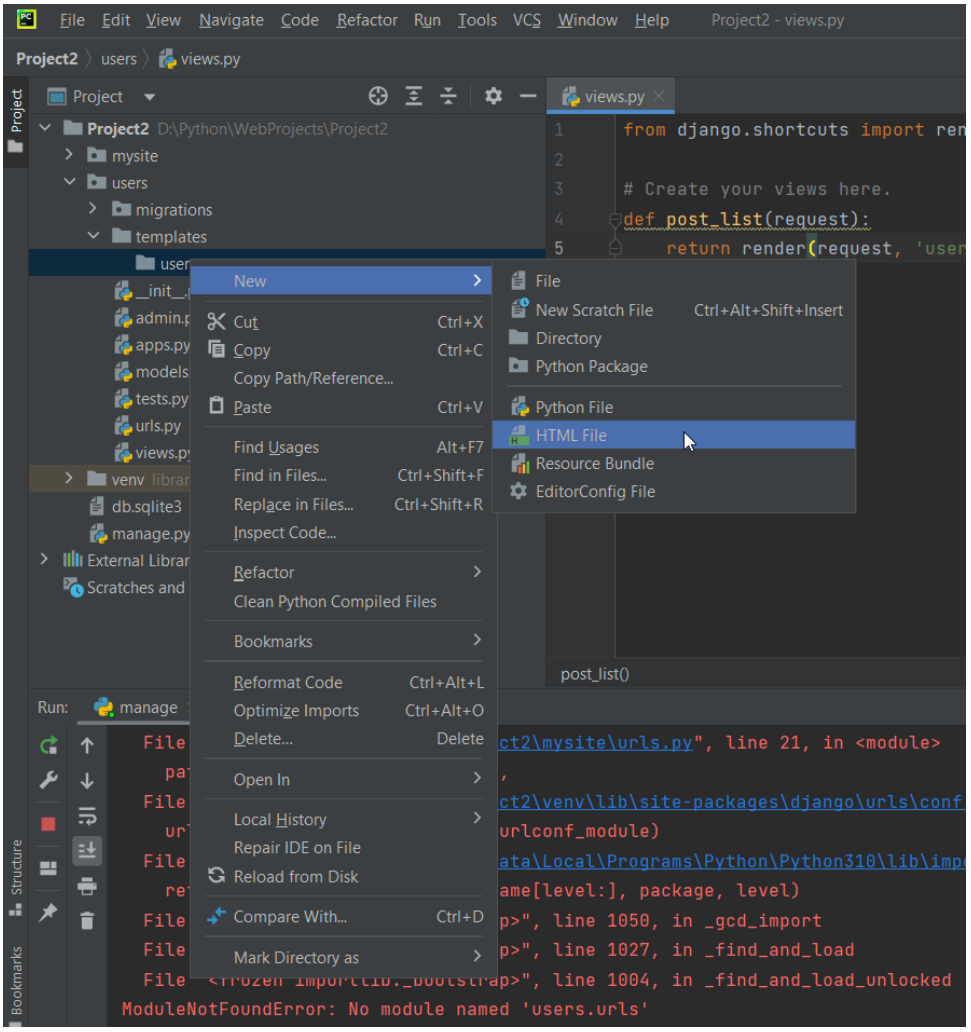


Получим следующее дерево:



В конечной папке users создаем два html-файла:

`users/templates/users/index.html`
`users/templates/users/results.html`



Файл index.html должен содержать следующее:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Django + MySQL</title>
</head>
<body>

<h1>Users</h1>
<p>

<form action="" method="post">
   <fieldset id="search">
       <legend>Data for searching User by name</legend>
```

```
        {% csrf_token %}
        {{ form }}
        <input type="submit" value="Search">
    </fieldset>
</form>

</body>
</html>
```

Обратите внимание на строку {{form}} – это как раз наша форма описанная в Django, у нее было такое имя.

Почти все остальное обычные теги HTML. В теге создания формы <form> указан метод веб-запроса: method = POST, запрос именно такого типа мы ожидаем в get_name().

Файл results.html, который будет выводить результаты поиска, заполняем следующим кодом:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Django + MySQL</title>
</head>
<body>

<h1>Users</h1>
<p>

<div>
    <fieldset>
        <legend>Searching by: {{substring}}</legend>

        {% for u in user %}
        {{u}} <br>
        {% endfor %}

    </fieldset>
```

```
</div>

</body>
</html>
```

Тут мы получили список данных по найденным пользователям и выводим данные в цикле.

Наше веб-приложение готово.

3.7 Запуск web-приложения

Запустите web-сервер, как это делалось в пункте 3.3, не забудьте перед этим запустить Docker и контейнер my-mysql.