Создание веб сайтов на Python Django

Django — это свободный фреймворк для веб-приложений на языке Python, использующий шаблон проектирования MVC.

Шаблон MVC (Model-View-Controller) — это разделение архитектуры приложения на три модуля: модель (Model), представление (View) и контроллер (Controller). Он позволяет изменять компоненты независимо друг от друга для простой разработки и поддержки веб-приложений.

Для начала установим виртуальное окружение в наш проект, для стабильной работы библиотек. Виртуальное окружение - это инструмент, позволяющий создавать изолированные среды для выполнения и разработки приложений. Делается это в терминале при помощи команды «python -m venv {имя}»



После скачивания в нашей директории появляется папка с введённым именем:



Теперь чтобы активировать виртуальное окружение нужно ввести следующую инструкцию в терминал директории:



Где venv - это имя вашего виртуального окружения. Если всё сделано правильно, то в терминале перед путём к директории появится имя виртуального окружения:



Теперь можно устанавливать нужные нам библиотеки. Делать это мы будем при помощи пакетного менеджера “pip”. Для начала установим библиотеку Django. Чтобы сделать это нужно ввести в терминале нашего проекта следующую инструкцию:



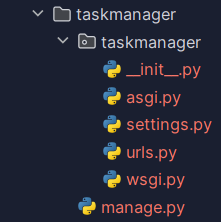
Теперь в нашу среду выполнения изолированно от глобальной среды установилась библиотека Django. Теперь в первую очередь что мы сделаем - это создадим в нашей директории Django-проект. Для этого нужно ввести в терминале следующую инструкцию:



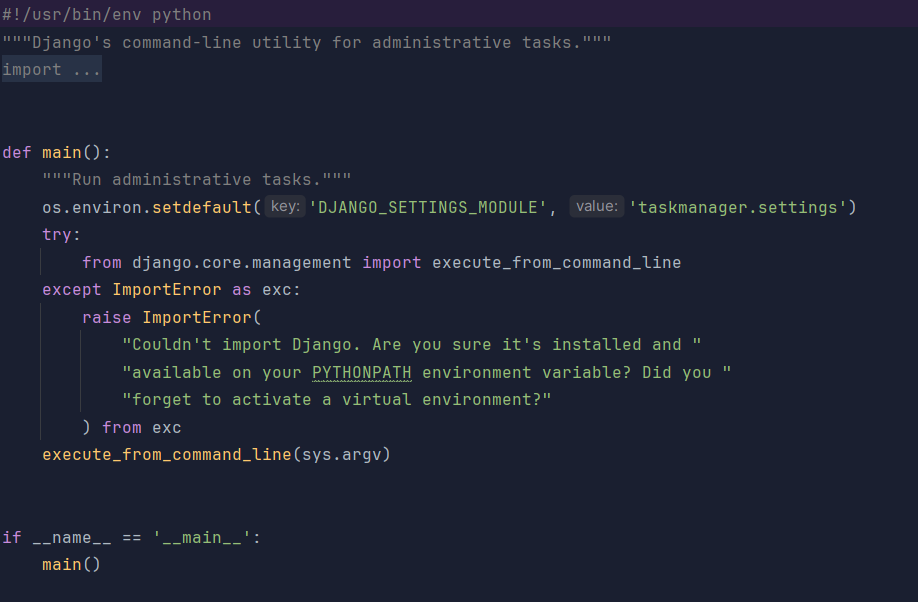
Где “taskmanager” - это имя, которым мы желаем назвать проект. После выполнения данной инструкции в директории нашего проекта появится папка проекта:



Теперь рассмотрим конфигурацию проекта:

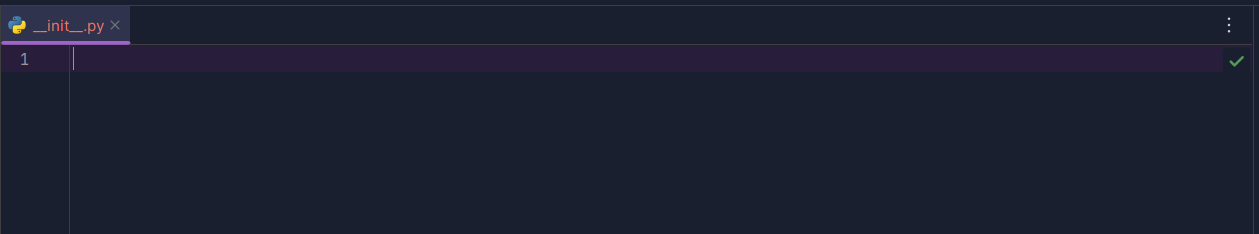


Для начала рассмотрим такой файл как “manage.py”. Он содержит следующий код:



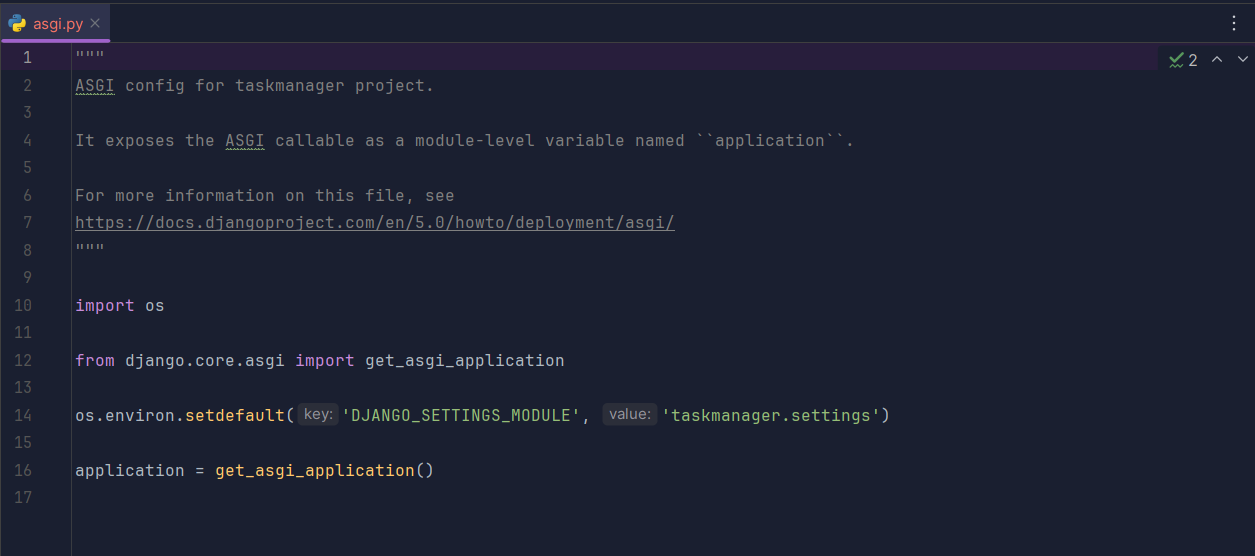
Напрямую работать мы с ним не будем, однако будем довольно часто взаимодействовать с ним при помощи терминала. При помощи него мы можем создавать локальный сервер, проводить какие-либо миграции и так далее.

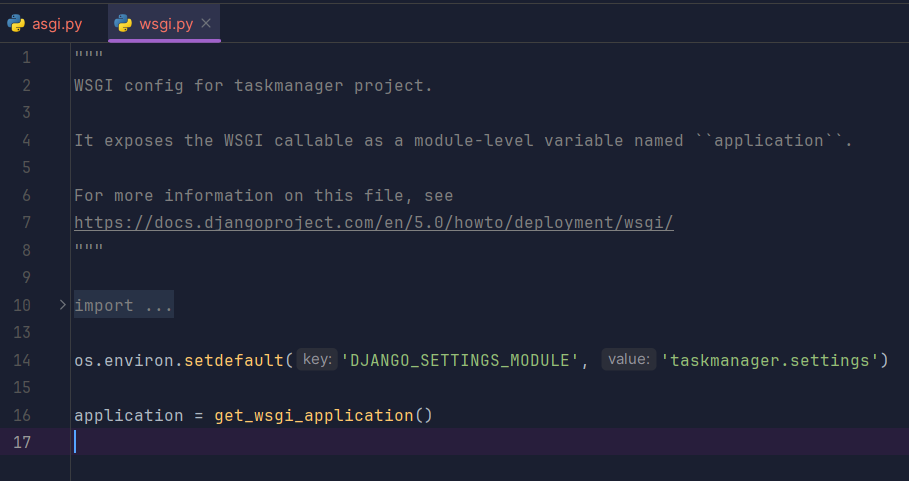
Далее рассмотрим такой файл как “\_\_init\_\_.py”:



Изначально он у нас пустой, но в будущем в этот файл можно записывать какие-либо характеристики, которые в дальнейшем нужно будет обрабатывать. Однако зачастую он всё таки остаётся пустым.

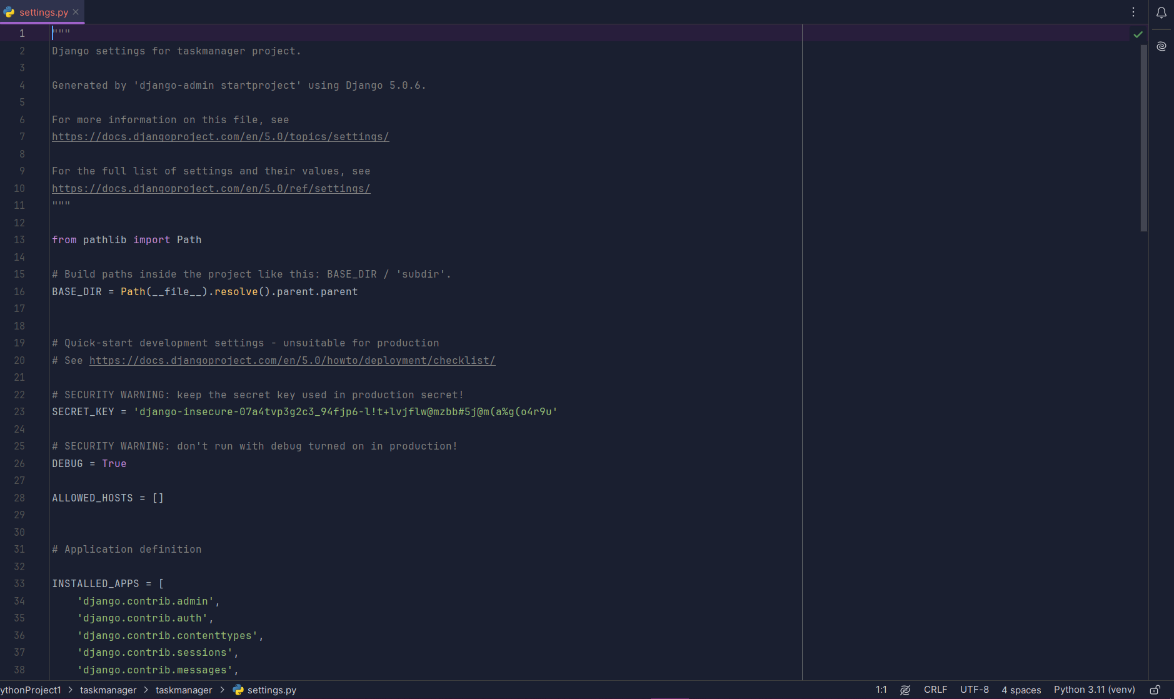
Следующие на рассмотрении файлы “asgi.py/wsgi.py”:





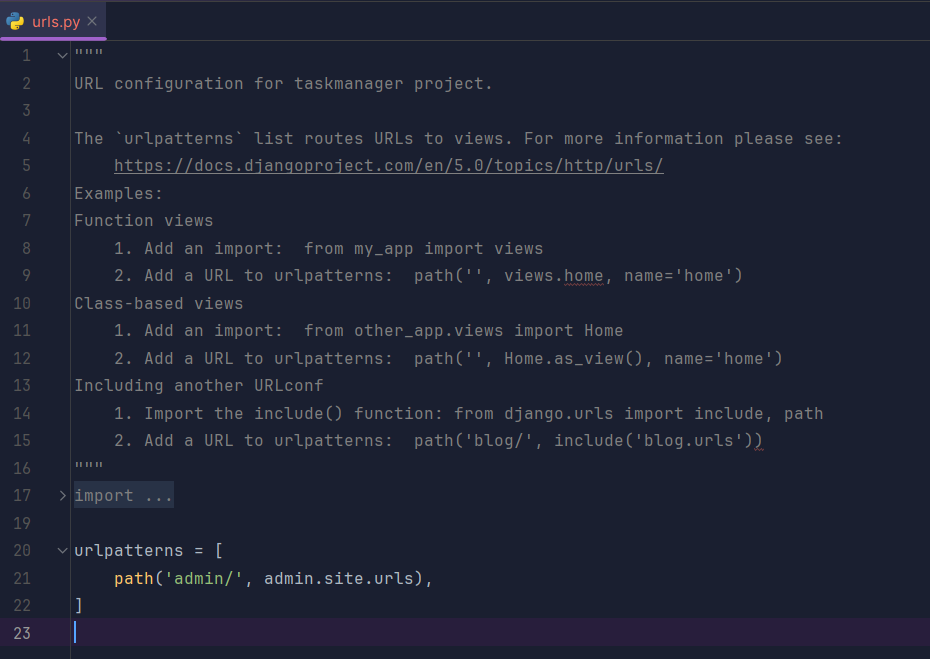
С ними мы также работаем лишь косвенно, не изменяя внутренний код. Они служат для установления связи между сервером и нашим проектом. По сути функционал у них одинаковый и отличаются они тем, что “asgi.py” имеет более новый стандарт.

Далее идёт файл “setting.py”:



Данный файл хранит в себе все глобальные настройки нашего проекта. По ходу дальнейшего развития проекта мы будем вносить в него некоторые изменения.

И последним идёт файл “urls.py”:



Данный файл служит для отслеживания url-адресов. Сейчас код в данном файле при переходе на страницу “admin” открывает панель администратора. Панель администратора по умолчанию устанавливается на абсолютно любой проект Django.

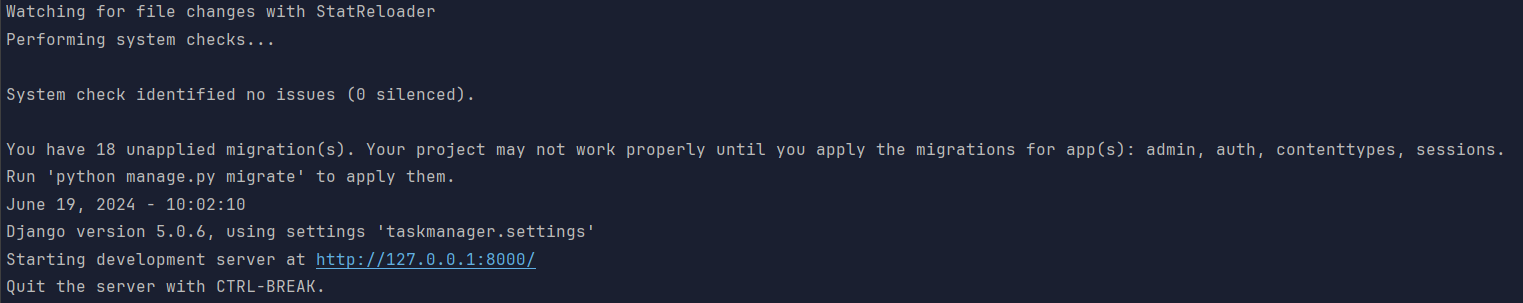
С разбором конфигурации проекта разобрались. Теперь можно попробовать запустить локальный сервер и посмотреть как изначально выглядит наш проект. Для этого нам нужно для начала перейти в папку, где находится файл “manage.py”:



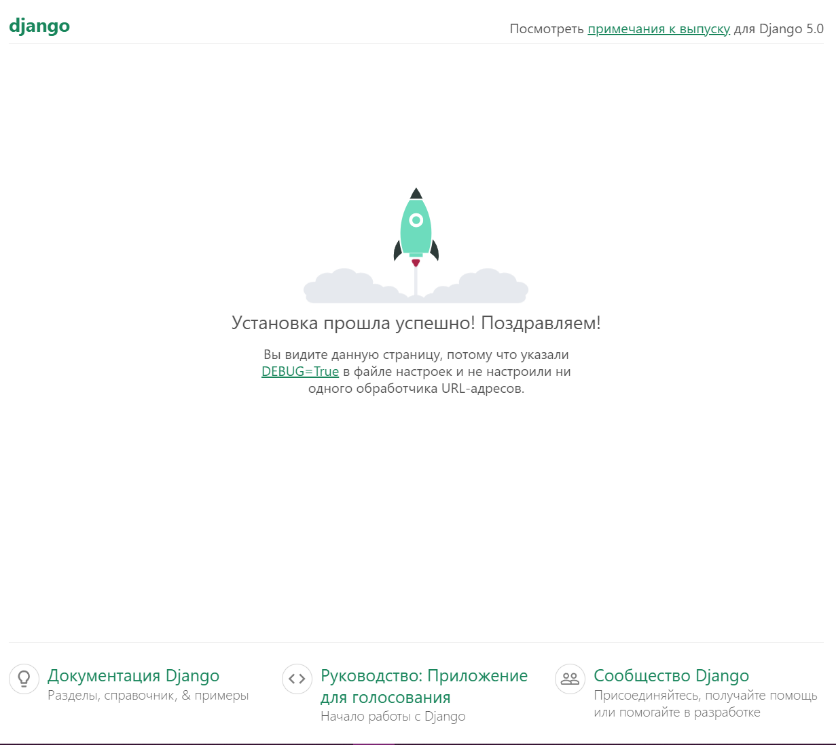
И теперь, когда мы находимся в папке нашего проекта мы можем запустить локальный сервер при помощи следующей команды:



После выполнения данной команды мы в терминале получаем IP-адрес по которому запущен наш локальный сервер:



Перейдя по ссылке мы видим следующее:

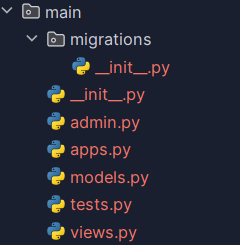


Данная страница свидетельствует о успешном запуске проекта и появилась она так как в нашем проекте пока нет ни одного HTML - шаблона, на который мы могли бы ссылаться. Чтобы остановить локальный сервер мы можем в терминале просто нажать сочетание клавиш “Ctrl + C”.

Теперь, когда мы разобрались с основным Django-проектом мы можем создать собственное приложение внутри него. Приложение в Django это какая-либо выделенная структура. Например если мы хотим создать на сайте форум, то проще вывести его в отдельную категорию и работать с ним. Сколько их создавать мы решаем сами, главное, чтобы оно было хотя-бы одно. Мы можем все категории поместить в одно приложении, или же для каждой категории мы можем создать отдельные приложения. Для того, чтобы создать приложение мы должны в терминале ввести следующую инструкцию:



Где “main” - это имя, которым мы хотим назвать наше приложение. После выполнения данного кода в директории нашего проекта появится приложение с указанным именем:

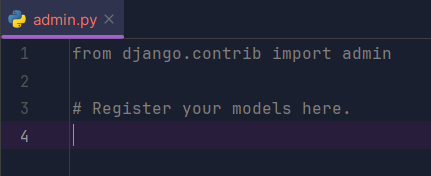


Оно также имеет свою конфигурацию, с которой мы в дальнейшем будем взаимодействовать.

В любом приложении есть папка “migration” с файлом “\_\_init\_\_.py”. Она нужна для отслеживания миграций на нашем проекте. Подробнее их мы рассмотрим позже, но отметим, что нужны они для взаимодействия с базами данных.

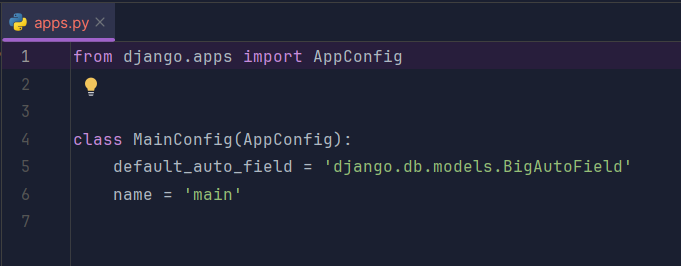
У файла “\_\_init\_\_.py”, находящегося внутри приложения такой же функционал как и у всех других одноимённых папок. Изначально она пустая и записывать в неё мы будем какие-либо характеристики передаваемые сразу при запуске проекта.

Дальше идёт файл “admin.py”:



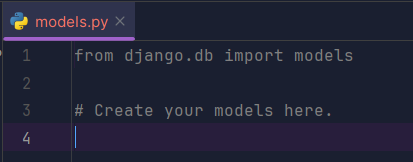
В нём мы будем описывать конкретную структуру панели администратора.

Следующий файл “apps.py”:



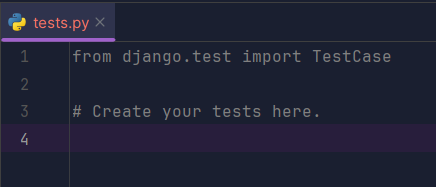
В этом файле мы можем описать какие-либо глобальные настройки, которые принадлежат конкретному этому приложению.

Затем идёт файл “models.py”:



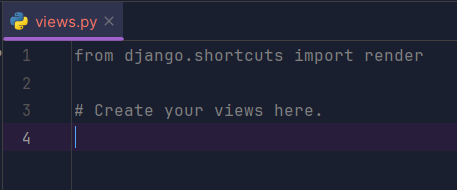
В нём мы будем создавать модели по которым будут строиться таблицы в базе данных.

В файле “tests.py”:



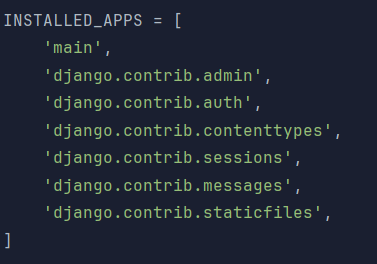
Мы можем создавать uni-тесты для тестирования работы нашего приложения.

И последним на очереди файл “views.py”:



В данном файле мы можем прописать методы, которые будут вызываться при переходе пользователя на какие-либо страницы.

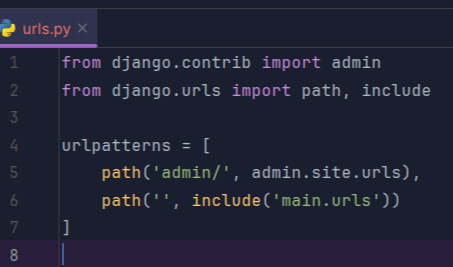
После того, как мы создали какое-либо приложение обязательно нужно в глобальных настройках нашего проекта добавить его в список “INSTALLED\_APPS”:



Теперь Django будет понимать, что внутри нашего проекта есть такое приложение как “main”.

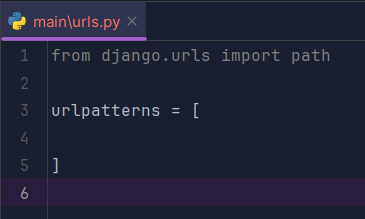
После того, как мы рассмотрели конфигурацию нашего проекта и приложения перейдём к непосредственно работе с нашим проектом. Для начала научимся отслеживать некоторые URL-адреса. В файле «urls.py» уже есть прописано отслеживание перехода на страницу “admin”. Добавим также отслеживание перехода на главную страницу:

Для этого нужно добавить вызов ещё одного метода “path” и передать ему в качестве аргументов страницу, которую мы хотим отследить и имя файла “urls” внутри нашего приложения. Чтобы сделать это мы импортируем также библиотеку “include” и пишем следующий код:



И также создаём файл “urls.py” в нашем приложении. Делается это для того, чтобы отслеживание url-адресов относимых к определённой категории отслеживало приложение этой категории.

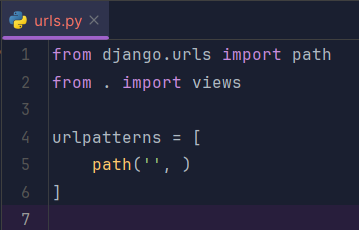
После создания файла “urls.py” внутри нашего приложения мы помещаем в него следующий код:



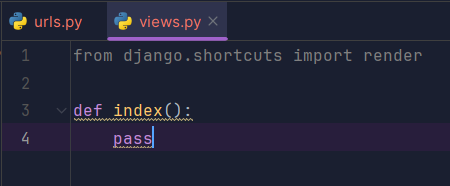
И теперь логика нашего проекта такова:

Когда пользователь заходит на главную страницу нашего сайта вызывается файл “urls.py” приналдежаший общему проекту taskmanager. А внутри проекта при помощи метода “path” вызывается файл “urls.py”, который принадлежит уже приложению “main”, в котором мы будем отслеживать переходы по url-адресам принадлежащих категории этого приложения.

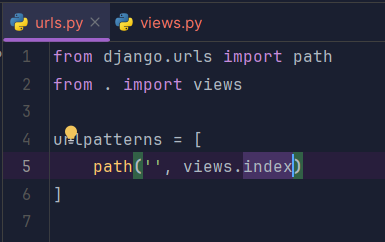
Теперь для отслеживания url-адресов мы внутри файла “main\urls.py” вызываем метод “path”, которому передаём отслеживаемый url-адрес и определённый метод из файла “views.py”. Чтобы это сделать импортируем этот файл внутрь файла “urls.py”.



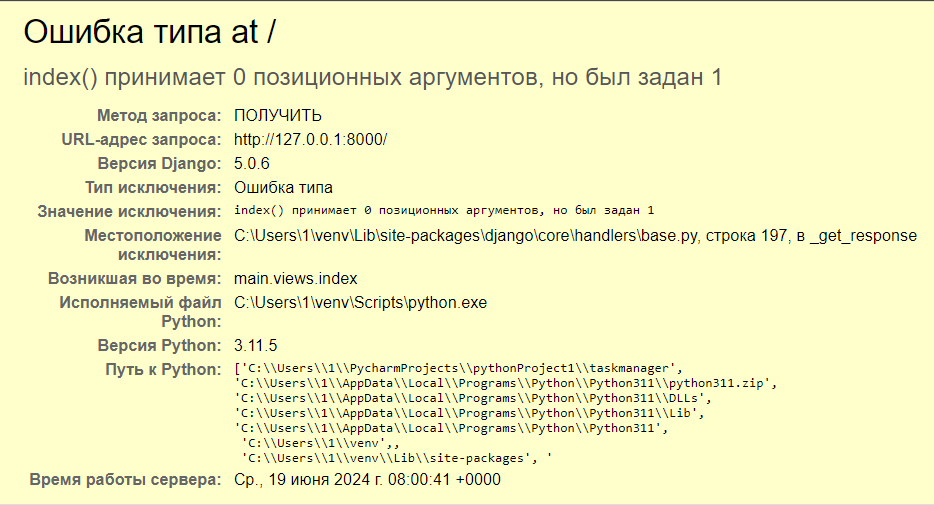
Дальше внутри файла “views.py” мы прописываем какой-либо метод:



Теперь мы можем вызвать его в файле “urls.py”:

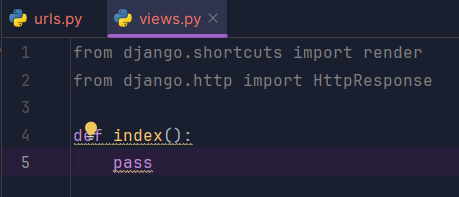


Причём круглые скобки после вызова метода писать не надо. Всё, теперь мы отслеживаем переход на главную страницу. Если мы запустим локальный сервер, то получим следующее:

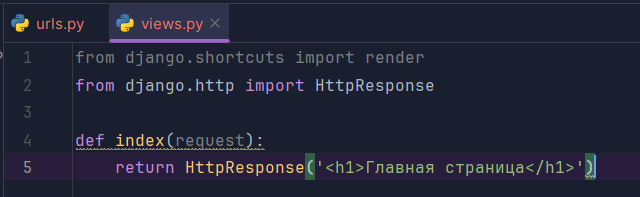


Это произошло так как метод “index” не имеет никакого функционала. Чтобы это исправить нужно будет указать как принимаемый аргумент “request”.

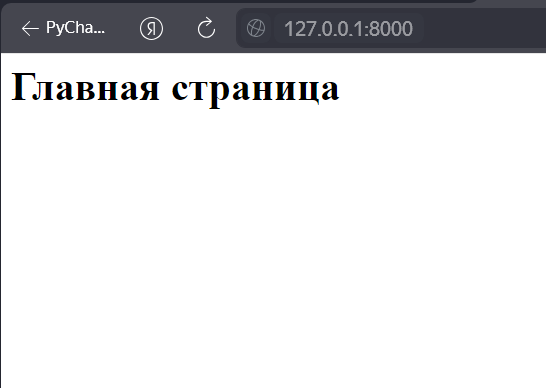
Пока будем выводить обычный текст, без никакого HTML-шаблона. Для этого нам нужно в файле “views.py” кое что импортировать, а именно из папки django.http импортируем класс HttpResponse:



И чтобы вывести какой-либо текст напишем следующий код:

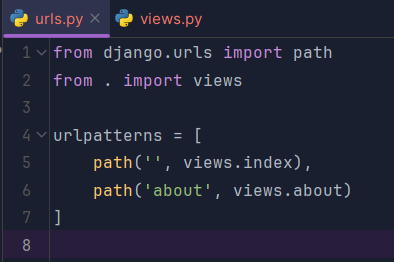


Теперь при запуске локального сервера нас будет встречать такая надпись:

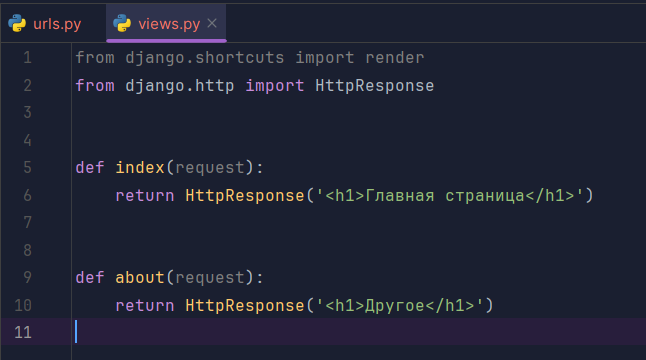


Всё отлично работает.

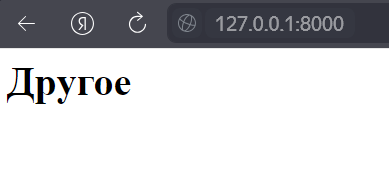
Также отследим переход на ещё одну страницу, пусть это будет страница “about”. Для этого в файле “main\urls.py” добавим отслеживание ещё одной страницы и передадим метод, который будет выполняться при входе на страницу, этот метод мы также создадим в файле “views.py”:



Создадим метод:



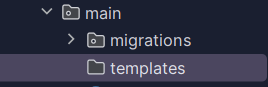
Теперь при переходе на страницу “about” мы увидим следующее:



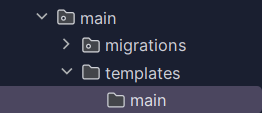
Всё работает.

Пока мы передаём только какой-то текст при помощи функции HttpResponce, однако при помощи неё нельзя передать целый HTML-шаблон. А это мы сделаем иначе.

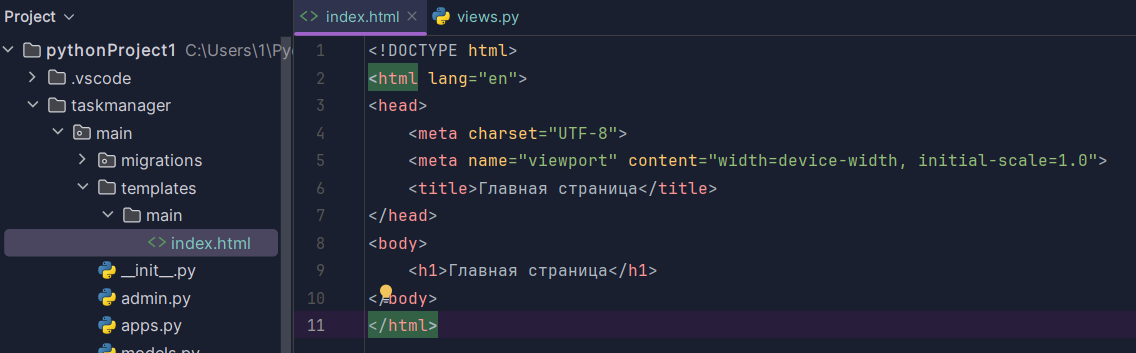
Для начала нужно в папке приложения создать папку под названием “templates”. Название папки должно быть именно таким в соответствие с документацией фреймворка.



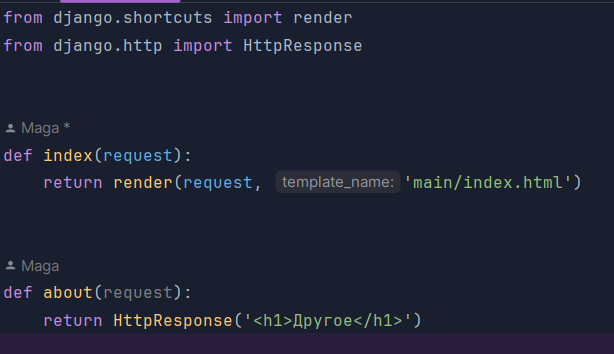
И теперь согласно документации внутри папки “templates” рекомендуется создать ещё одну папку с названием приложения в котором она находится. Это нужно сделать так как папок “templates” в проекте может быть много. И чтобы не было конфликтов мы также создаём внутри этой папки ещё одну с именем приложения.



Ну и теперь мы можем поместить в эту папку HTML-шаблон.

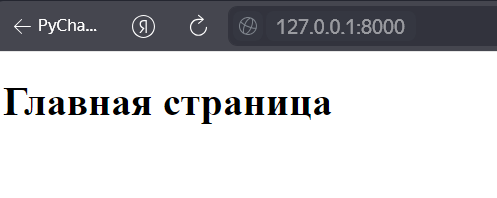


Теперь мы сможем вызвать этот шаблон в методах, которые вызываются при отслеживании url-адресов. Но для начала нам нужно кое-что поменять. Так как функция HttpResponse служит для передачи строки она не подойдёт для передачи шаблона. Поэтому вместо неё мы используем функцию “render”.



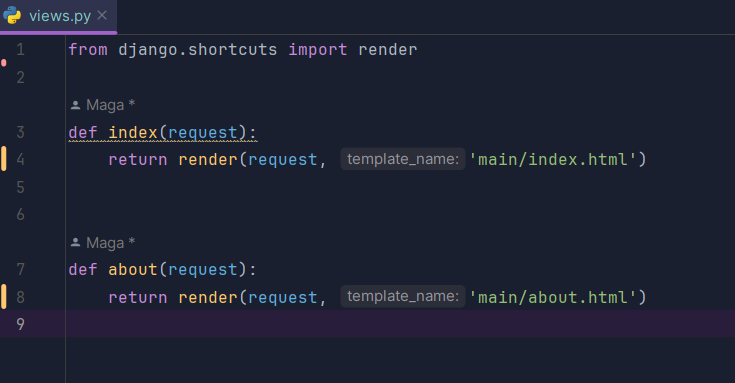
Мы вызываем метод “render” и в качестве аргументов передаём “request” и путь к нашему HTML-шаблону.

Запустим локальный сервер:

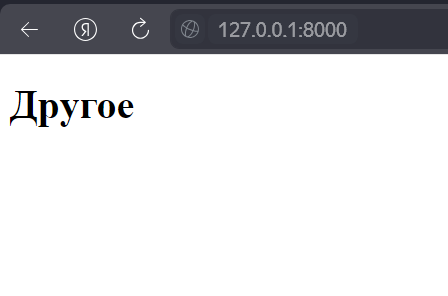
****

Всё исправно работает.

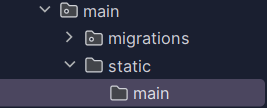
Теперь сделаем тоже самое со страницей “about”



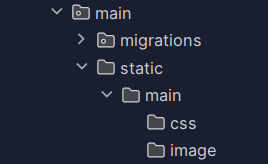
При запуске локального сервера всё также отлично отображается



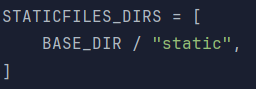
С HTML-шаблонами разобрались, но может появиться нужда также сохранять css или js файлы. Для этого внутри приложения также нужно создать папку с определённым названием, а именно “static” и точно также как и в папке “templates” создать внутри ещё одну папку с именем приложения. Она будет хранить в себе все статические файлы включая и изображения, аудио, видео и так далее.



И также мы создадим внутри папки “main/static/main” папки для упорядочивания файлов по типам. Пока это будет “css” и “image”.



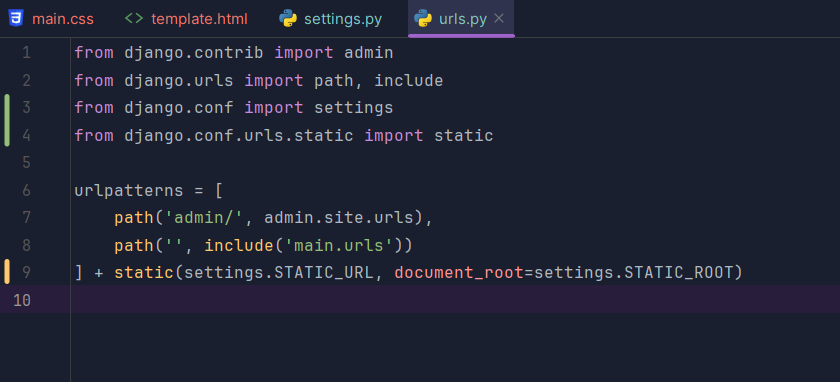
И также для корректной работы статических файлов согласно документации Django, нам нужно добавить в файл “settings.py” следующую конструкцию



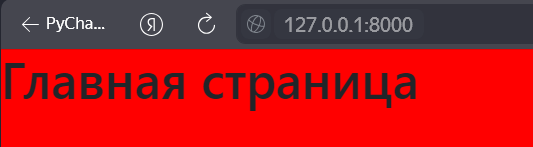
И ещё в папку “taskmanager/urls.py” добавить следующую конструкцию



В конечном итоге выглядеть это будет так



Теперь если мы подключим css-файл к нашему HTML-шаблону всё будет корректно работать.



Я просто поменял цвет фона на красный

