

Урок 8. Подключаем статику

ОГЛАВЛЕНИЕ

[что такое статика 3](#_Toc78473009)

[дорабатываем шаблонизатор 4](#_Toc78473010)

[дорабатываем фреймворк 5](#_Toc78473011)

[настраиваем файл settings.py 12](#_Toc78473012)

[подключаем статику 14](#_Toc78473013)

[Заключение 18](#_Toc78473014)



# что такое статика

Если вы уже создавали проекты на Django, то точно знаете, что в каждом проекте существовала папка static, в которой хранились еще несколько директорий: css, images, js, fonts. А уже в этих директориях находились служебные файлы, называемые статическими. Это файлы стилей, js-скрипты, изображения. Они нужны нашему веб-приложению, но в них мы не передаем какого-либо контекста, поэтому данные файлы называются статическими или просто статикой.

Наверняка в процессе прохождения курса вы уже задумывались, почему мы не подключаем статику, т.е. стилизацию задаем непосредственно через HTML, не добавляем картинки в наше веб-приложение. Возможно вы даже пытались все это добавить, но ничего не работало. Потому возможность подключения статики обеспечивает нам фреймворк, но мы его еще не научили этому, а значит вернемся к доработке фреймворка, а затем доработаем наше веб-приложение.

# дорабатываем шаблонизатор

Точнее мы будем дорабатывать не сам шаблонизатор (мы ведь подключили и используем jinja2), а функцию **render()**, которую реализовали в модуле **templator.py**.

**Листинг 1. step\_8/simba\_framework/templator.py**

**from** jinja2 **import** Environment, FileSystemLoader  
  
  
**def** render(template\_name, folder=**'templates'**,

static\_url=**'/static/'**, \*\*kwargs):  
env = Environment()  
 env.loader = FileSystemLoader(folder)  
 env.globals[**'static'**] = static\_url  
 template = env.get\_template(template\_name)  
 **return** template.render(\*\*kwargs)

Если вновь обратить внимание на Django-проекты, то там у нас был модуль **settings.py**, в котором мы прописывали константы, в частности, **STATIC\_URL**. Эта константа определяла общедоступный URL для статических файлов. Чаще всего значением этой константы было ‘/static/’. Благодаря этому каждый наш статический файл имел относительный URL ‘/static/new\_app/my\_image.jpg’ и сразу было понятно, что это статический файл нашего проекта.

При этом при добавлении статических файлов в шаблоны мы применяли необходимые теги шаблона для правильной привязки статики:

**{%** [**load**](https://djbook.ru/rel1.6/ref/templates/builtins.html#load) staticfiles **%}**

<img src="**{%** [**static**](https://djbook.ru/rel1.6/ref/templates/builtins.html#static) "new\_app/my\_image.jpg" **%}**" alt="my image"/>

В нашем случае общедоступный URL мы пропишем как аргумент функции **render()** – static\_url='/static/'**.**

Нам нужно, чтобы этот URL автоматически добавлялся к ссылкам наших статических файлов. А эти ссылки, как вы помните, находятся у нас в HTML-шаблонах.

Для настройки добавления URL нам необходимо вновь вспомнить о классе **Environment()**, который соответствует среде окружения шаблонизатора jinja2. В среде окружения находятся, в частности, глобальные переменные, различные параметры и конфигурации.

В данном случае нам нужен атрибут **Environment.globals**, представляющий собой словарь глобальных переменных. Эти переменные доступны в шаблоне постоянно. Именно это нам и нужно. Добавим в этот словарь новый элемент – ссылку на общедоступный URL – ‘/static/’.

env.globals[**'static'**] = static\_url

В остальном код функции render() у нас не изменился.

# дорабатываем фреймворк

Теперь нужно научить наш фреймворк тому, чтобы он понимал, что, встречая на своем пути при рендеринге шаблонов вот такие адреса: ‘/static/new\_app/my\_image.jpg’, нужно «доставать» из папки со статикой нужный файл и обеспечивать его отображение.

Наш фреймворк научился понимать, что ему делать с различными адресами на выражении:

**if** path **in** self.routes\_lst:

….

Оно находится в файле **main.py**.

Таким образом, в более ранних уроках реализовали запуск соответствующего каждому адресу обработчика.

Метод **\_\_call\_\_()** нашего главного класса-фреймворка теперь будет выглядеть так:

**Листинг 2. step\_8/simba\_framework/main.py**

**class** Framework:  
  
 *"""Класс Framework - основа WSGI-фреймворка"""* **def** \_\_init\_\_(self, settings, routes\_obj):  
 self.routes\_lst = routes\_obj  
 self.settings = settings  
  
 **def** \_\_call\_\_(self, environ, start\_response):  
 *# Получаем адрес, по которому пользователь выполнил переход* path = environ[**'PATH\_INFO'**]  
  
 *# Добавляем закрывающий слеш* **if not** path.endswith(**'/'**):  
 path = **f'{**path**}/'** request = {}  
 *# Получаем все данные запроса* method = environ[**'REQUEST\_METHOD'**]  
 request[**'method'**] = method  
  
 **if** method == **'POST'**:  
 data = PostRequests().get\_request\_params(environ)  
 request[**'data'**] = data  
 print(**f'Нам пришёл post-запрос:**

**{**Framework.decode\_value(data)**}'**)  
 **if** method == **'GET'**:  
 request\_params = GetRequests().get\_request\_params(environ)  
 request[**'request\_params'**] = request\_params  
 print(**f'Нам пришли GET-параметры: {**request\_params**}'**)  
  
 *# Находим нужный контроллер* **if** path **in** self.routes\_lst:  
 view = self.routes\_lst[path]  
 content\_type = self.get\_content\_type(path)  
 code, body = view(request)  
 body = body.encode(**'utf-8'**)  
  
 **elif** path.startswith(self.settings.STATIC\_URL):  
 *# /static/img/logo.jpg/ -> img/logo.jpg* file\_path = path[len(self.settings.STATIC\_URL):len(path)-1]  
 content\_type = self.get\_content\_type(file\_path)  
 code, body = self.get\_static(self.settings.STATIC\_FILES\_DIR,  
 file\_path)  
  
 **else**:  
 view = PageNotFound404()  
 content\_type = self.get\_content\_type(path)  
 code, body = view(request)  
 body = body.encode(**'utf-8'**)  
 start\_response(code, [(**'Content-Type'**, content\_type)])  
  
 **return** [body]

Выполним подробный разбор кода:

**if** path **in** self.routes\_lst:

view = self.routes\_lst[path]

content\_type = self.get\_content\_type(path)

code, body = view(request)

body = body.encode(**'utf-8'**)

Выражения этого блока нам хорошо знакомы. Мы извлекаем объект обработчика, опираясь на текущий адрес-путь (**path**).

А вот дальше новое выражение:

content\_type = self.get\_content\_type(path)

Давайте вспомним, ранее в коде нашего фреймворка была такая команда:

start\_response(code, [(**'Content-Type'**, **'text/html'**)])

Мы отправляли код HTTP-ответа и HTTP-заголовок. Заголовок необходим для корректной адаптации сервером ответа. Когда мы возвращали только тело HTML-страницы, то прописывали значение HTTP-заголовка явно. Мы указывали:

[(**'Content-Type'**, **'text/html'**)]

Теперь же нам нужно возвращать в качестве тела ответа и статический объект, а потом возможно и медиа-файл. Чтобы не создавать многочисленный хардкод давайте сделаем так. Создадим в папке **components** модуль **content\_types.py**.

**Листинг 3. step\_8/components/content\_types.py**

CONTENT\_TYPES\_MAP = {  
 **".3gp"**: **"video/3gpp"**,  
 **".3gpp"**: **"video/3gpp"**,  
 **".7z"**: **"application/x-7z-compressed"**,  
 **".ai"**: **"application/postscript"**,  
 **".asf"**: **"video/x-ms-asf"**,  
 **".asx"**: **"video/x-ms-asf"**,  
 **".atom"**: **"application/atom+xml"**,  
 **".avi"**: **"video/x-msvideo"**,  
 **".bmp"**: **"image/x-ms-bmp"**,  
 **".cco"**: **"application/x-cocoa"**,  
 **".crt"**: **"application/x-x509-ca-cert"**,  
 **".css"**: **"text/css"**,  
 **".der"**: **"application/x-x509-ca-cert"**,  
 **".doc"**: **"application/msword"**,  
 **".docx"**: **"application/vnd.openxmlformats-**

**officedocument.wordprocessingml.document"**,  
 **".ear"**: **"application/java-archive"**,  
 **".eot"**: **"application/vnd.ms-fontobject"**,  
 **".eps"**: **"application/postscript"**,  
 **".flv"**: **"video/x-flv"**,  
 **".gif"**: **"image/gif"**,  
 **".hqx"**: **"application/mac-binhex40"**,  
 **".htc"**: **"text/x-component"**,  
 **".htm"**: **"text/html"**,  
 **".html"**: **"text/html"**,  
 **".ico"**: **"image/x-icon"**,  
 **".jad"**: **"text/vnd.sun.j2me.app-descriptor"**,  
 **".jar"**: **"application/java-archive"**,  
 **".jardiff"**: **"application/x-java-archive-diff"**,  
 **".jng"**: **"image/x-jng"**,  
 **".jnlp"**: **"application/x-java-jnlp-file"**,  
 **".jpeg"**: **"image/jpeg"**,  
 **".jpg"**: **"image/jpeg"**,  
 **".js"**: **"text/javascript"**,  
 **".json"**: **"application/json"**,  
 **".kar"**: **"audio/midi"**,  
 **".kml"**: **"application/vnd.google-earth.kml+xml"**,  
 **".kmz"**: **"application/vnd.google-earth.kmz"**,  
 **".m3u8"**: **"application/vnd.apple.mpegurl"**,  
 **".m4a"**: **"audio/x-m4a"**,  
 **".m4v"**: **"video/x-m4v"**,  
 **".mid"**: **"audio/midi"**,  
 **".midi"**: **"audio/midi"**,  
 **".mjs"**: **"text/javascript"**,  
 **".mml"**: **"text/mathml"**,  
 **".mng"**: **"video/x-mng"**,  
 **".mov"**: **"video/quicktime"**,  
 **".mp3"**: **"audio/mpeg"**,  
 **".mp4"**: **"video/mp4"**,  
 **".mpeg"**: **"video/mpeg"**,  
 **".mpg"**: **"video/mpeg"**,  
 **".ogg"**: **"audio/ogg"**,  
 **".pdb"**: **"application/x-pilot"**,  
 **".pdf"**: **"application/pdf"**,  
 **".pem"**: **"application/x-x509-ca-cert"**,  
 **".pl"**: **"application/x-perl"**,  
 **".pm"**: **"application/x-perl"**,  
 **".png"**: **"image/png"**,  
 **".ppt"**: **"application/vnd.ms-powerpoint"**,  
 **".pptx"**: **"application/vnd.openxmlformats-**

**officedocument.presentationml.presentation"**,  
 **".prc"**: **"application/x-pilot"**,  
 **".ps"**: **"application/postscript"**,  
 **".ra"**: **"audio/x-realaudio"**,  
 **".rar"**: **"application/x-rar-compressed"**,  
 **".rpm"**: **"application/x-redhat-package-manager"**,  
 **".rss"**: **"application/rss+xml"**,  
 **".rtf"**: **"application/rtf"**,  
 **".run"**: **"application/x-makeself"**,  
 **".sea"**: **"application/x-sea"**,  
 **".shtml"**: **"text/html"**,  
 **".sit"**: **"application/x-stuffit"**,  
 **".svg"**: **"image/svg+xml"**,  
 **".svgz"**: **"image/svg+xml"**,  
 **".swf"**: **"application/x-shockwave-flash"**,  
 **".tcl"**: **"application/x-tcl"**,  
 **".tif"**: **"image/tiff"**,  
 **".tiff"**: **"image/tiff"**,  
 **".tk"**: **"application/x-tcl"**,  
 **".ts"**: **"video/mp2t"**,  
 **".txt"**: **"text/plain"**,  
 **".wasm"**: **"application/wasm"**,  
 **".war"**: **"application/java-archive"**,  
 **".wbmp"**: **"image/vnd.wap.wbmp"**,  
 **".webm"**: **"video/webm"**,  
 **".webp"**: **"image/webp"**,  
 **".wml"**: **"text/vnd.wap.wml"**,  
 **".wmlc"**: **"application/vnd.wap.wmlc"**,  
 **".wmv"**: **"video/x-ms-wmv"**,  
 **".woff"**: **"application/font-woff"**,  
 **".woff2"**: **"font/woff2"**,  
 **".xhtml"**: **"application/xhtml+xml"**,  
 **".xls"**: **"application/vnd.ms-excel"**,  
 **".xlsx"**: **"application/vnd.openxmlformats-**

**officedocument.spreadsheetml.sheet"**,  
 **".xml"**: **"text/xml"**,  
 **".xpi"**: **"application/x-xpinstall"**,  
 **".xspf"**: **"application/xspf+xml"**,  
 **".zip"**: **"application/zip"**,  
 **"apple-app-site-association"**: **"application/pkc7-mime"**,  
 **"crossdomain.xml"**: **"text/x-cross-domain-policy"**,  
 }

Вернемся к выражению:

content\_type = self.get\_content\_type(path)

Функцию **get\_content\_type()** мы добавим в качестве метода класса Framework:

**Листинг 4. step\_8/simba\_framework/main.py**

@staticmethod  
**def** get\_content\_type(file\_path, content\_types\_map=CONTENT\_TYPES\_MAP):  
 file\_name = path.basename(file\_path).lower()  
 extension = path.splitext(file\_name)[1]  
 **return** content\_types\_map.get(extension, **"text/html"**)

Сделаем разбор кода.

**def** get\_content\_type(file\_path, content\_types\_map=CONTENT\_TYPES\_MAP):

Функция принимает адрес страницы или объекта на странице, а также описанный выше словарь с HTTP-заголовками.

file\_name = path.basename(file\_path).lower()

Получаем имя файлы из переданного адреса, например, **styles.css**.

extension = path.splitext(file\_name)[1]

Извлекаем расширение файла, например, .css.

**return** content\_types\_map.get(extension, **"text/html"**)

Опираясь на полученное расширение извлекаем из словаря **content\_types\_map** нужный заголовок или, если ничего подходящего нет, возвращаем **"text/html".**

Теперь самое важное. Добавим в код метода **\_\_call\_\_()** еще одну ветвь. Она необходима для обработки путей к статическим файлам.

**Листинг 4. step\_8/simba\_framework/main.py**

**elif** path.startswith(self.settings.STATIC\_URL):  
 *# /static/images/logo.jpg/ -> images/logo.jpg* file\_path = path[len(self.settings.STATIC\_URL):len(path)-1]  
 content\_type = self.get\_content\_type(file\_path)  
 code, body = self.get\_static(self.settings.STATIC\_FILES\_DIR,  
 file\_path)

Сделаем разбор кода:

**elif** path.startswith(self.settings.STATIC\_URL):

Если путь начинается со ‘**/static/**‘(это выражение находится в переменной STATIC\_URL модуля **settings.py** – о нем мы поговорим ниже), тогда:

file\_path = path[len(self.settings.STATIC\_URL):len(path)-1]

Получаем путь до статического файла. Нам нужны адреса наподобие

‘/static/images/logo.jpg/’ привести к ‘images/logo.jpg’. Что мы и делаем с помощью данного выражения.

content\_type = self.get\_content\_type(file\_path)

Знакомое выражение. Возвращает нам нужный HTTP-заголовок по расширению файла, например, для jpg-файла.

А вот теперь очень важное выражение:

code, body = self.get\_static(self.settings.STATIC\_FILES\_DIR,  
 file\_path)

Функция **get\_static()** должна вернуть нам код и тело ответа в байтах. Эта функция принимает путь до директории со статическими файлами. В нашем случае это будет директория **staticfiles**, расположенная в корне проекта. Но указание, что это именно директория **staticfiles** будет находиться в переменной **STATIC\_FILES\_DIR** модуля **settings.py**.

Итак, добавим функцию **get\_static()** в качестве метода класса **Framework**.

**Листинг 5. step\_8/simba\_framework/main.py**

@staticmethod  
**def** get\_static(static\_dir, file\_path):  
 path\_to\_file = path.join(static\_dir, file\_path)  
 **with** open(path\_to\_file, **'rb'**) **as** f:  
 file\_content = f.read()  
 status\_code = **'200 OK'  
 return** status\_code, file\_content

Разберем код функции.

**def** get\_static(static\_dir, file\_path):

Принимает путь до директории со статикой (например, ‘step\_8\_3/staticfiles’), а также путь до статического файла (например, ‘images/logo.jpg’).

path\_to\_file = path.join(static\_dir, file\_path)

Выполняем сцепление директории и пути.

**with** open(path\_to\_file, **'rb'**) **as** f:  
 file\_content = f.read()

Открываем файл на чтение в байтовом режим и считываем его содержимое в переменную file\_content.

status\_code = **'200 OK'**

Формируем код ответа.

**return** status\_code, file\_content

Отправляем код и тело ответа.

Далее в файле **main.py** можно заметить, что мы еще немного перестроили логику обработки страницы 404. Логика осталась такая же, только немного перестроили код:

**Листинг 6. step\_8/simba\_framework/main.py**

**else**:  
 view = PageNotFound404()  
 content\_type = self.get\_content\_type(path)  
 code, body = view(request)  
 body = body.encode(**'utf-8'**)

Завершаем разбор функции **\_\_call\_\_()** нашего класса Framework.

start\_response(code, [(**'Content-Type'**, content\_type)])  
  
**return** [body]

Выполняем запуск обработчика запроса и возврат тела ответа в байтах.

# настраиваем файл settings.py

Создадим этот файл в директории components.

**Листинг 7. step\_8/components/settings.py**

**from** os **import** path  
**from** pathlib **import** Path  
  
ROOT\_DIR = Path(\_\_file\_\_).resolve().parent.parent  
STATIC\_FILES\_DIR = path.join(ROOT\_DIR, **'staticfiles'**)STATIC\_URL = **'/static/'**

ROOT\_DIR = Path(\_\_file\_\_).resolve().parent.parent

Получаем путь до корня проекта, например, step\_8\_3.

STATIC\_FILES\_DIR = path.join(ROOT\_DIR, **'staticfiles'**)

Добавляем к корню путь до директории со статикой.

STATIC\_URL = **'/static/'**

Определяем общую URL-часть всех статических файлов. Теперь все URL-адреса, начинающиеся со STATIC\_URL, рассматриваются как запросы статических файлов.

Теперь сделаем так, что наш фреймворк будет принимать этот файл **settings.py**. Добавим в модуль **main.py**:

**class** Framework:  
  
 *"""Класс Framework - основа WSGI-фреймворка"""* **def** \_\_init\_\_(self, settings, routes\_obj):

А передавать этот файл **settings.py** мы будем при запуске нашего проекта, а именно при создании объекта нашего класса-фреймворка:

Файл **run.py** теперь будет выглядеть так:

**Листинг 7. step\_8/run.py**

**from** wsgiref.simple\_server **import** make\_server  
**from** simba\_framework.main **import** Framework  
**from** views **import** routes  
**from** components **import** settings  
  
*# Создаем объект WSGI-приложения*application = Framework(settings, routes)  
  
**with** make\_server(**''**, 8080, application) **as** httpd:  
 print(**"Запуск на порту 8080..."**)  
 httpd.serve\_forever()

Итак, мы реализовали возможность работы со статикой.

# подключаем статику

В рамках проверки работоспособности реализованного функционала попробуем настроить и подключить стилизацию через css-файл и добавим изображение на одну из страниц нашего веб-приложения.

Создадим в директории ‘staticfiles/css’ файл-таблицу стилей. Перенесем туда все стилевые правила, которые мы раньше определяли через HTML-разметку.

**Листинг 8. step\_8/staticfiles/css/style.css**

body {  
 font-family: Arial, Verdana, sans-serif;  
 font-size: 90%;  
 color: #666;  
 background-color: #f8f8f8;}  
li {  
 line-height: 1.6em;}  
table {  
 border-spacing: 0px;}  
th, td {  
 padding: 5px 30px 5px 10px;  
 border-spacing: 0px;  
 font-size: 90%;  
 margin: 0px;}  
th, td {  
 text-align: left;  
 background-color: #e0e9f0;  
 border-top: 1px solid #f1f8fe;  
 border-bottom: 1px solid #cbd2d8;  
 border-right: 1px solid #cbd2d8;}  
tr.head th {  
 color: #fff;  
 background-color: #90b4d6;  
 border-bottom: 2px solid #547ca0;  
 border-right: 1px solid #749abe;  
 border-top: 1px solid #90b4d6;  
 text-align: center;  
 text-shadow: -1px -1px 1px #666666;  
 letter-spacing: 0.15em;}  
td {  
 text-shadow: 1px 1px 1px #ffffff;}  
tr.even td, tr.even th {  
 background-color: #e8eff5;}  
tr.head th:first-child {  
 -webkit-border-top-left-radius: 5px;  
 -moz-border-radius-topleft: 5px;  
 border-top-left-radius: 5px;}  
tr.head th:last-child {  
 -webkit-border-top-right-radius: 5px;  
 -moz-border-radius-topright: 5px;  
 border-top-right-radius: 5px;}  
fieldset {  
 width: 310px;  
 margin-top: 20px;  
 border: 1px solid #d6d6d6;  
 background-color: #ffffff;  
 line-height: 1.6em;}  
legend {  
 font-style: italic;  
 color: #666666;}  
input[type=**"text"**] {  
 width: 120px;  
 border: 1px solid #d6d6d6;  
 padding: 2px;  
 outline: none;}  
input[type=**"text"**]:focus,  
input[type=**"text"**]:hover {  
 background-color: #d0e2f0;  
 border: 1px solid #999999;}  
input[type=**"submit"**] {  
 border: 1px solid #006633;  
 background-color: #009966;  
 color: #ffffff;  
 border-radius: 5px;  
 padding: 5px;  
 margin-top: 10px;}  
input[type=**"submit"**]:hover {  
 border: 1px solid #006633;  
 background-color: #00CC33;  
 color: #ffffff;  
 cursor: pointer;}  
.title {  
 float: left;  
 width: 160px;  
 clear: left;}  
.submit {  
 width: 310px;  
 text-align: right;}  
.menu {  
 */\*float: right;\*/* position: relative;  
 height: 50px;}  
.menu li {  
 */\*  
 Меню сайта:  
 а. Для всех ссылок меню задать определённый стиль (Цвет текста, размер шрифта, начертание шрифта и т.д.)  
 b. Убрать маркеры списка.  
 \*/* font-style: italic;  
 font-size: 20px;  
 list-style-type: none;  
 */\* li - блочный элемент, стоятся с столбик, сделаем их все в строку \*/* display: inline-block;  
 width: 180px;  
 margin: 0 auto;  
 text-align: center;}  
.menu li a {  
 */\*  
 цвет ссылок в меню  
 \*/* color: #8B00FF;  
 font-size: 30px;}  
content{  
 position: relative;}

Т.к. у нас один общий файл стилизации, добавим ссылку на таблицу стилей в базовый шаблон:

**Листинг 9. step\_8/templates/base.html**

<!DOCTYPE **html**>  
<**html lang="en"**>  
<**head**>  
 <**meta charset="UTF-8"**>  
 <**title**>{% block title %}{% endblock %}</**title**>  
 <**link rel="stylesheet" href="{{ static }}css/style.css"**>  
</**head**>  
<**body**>  
 {% block menu %} {% endblock %}  
 {% block body %} {% endblock %}  
</**body**>  
</**html**>

Теперь нужно из каждого шаблона-наследника и всех подшаблонов убрать ранее используемые выражения:

{% block style %}  
 {% include "inc-style.html" %}  
{% endblock %}

И можно удалить сам файл **inc-style.html**.

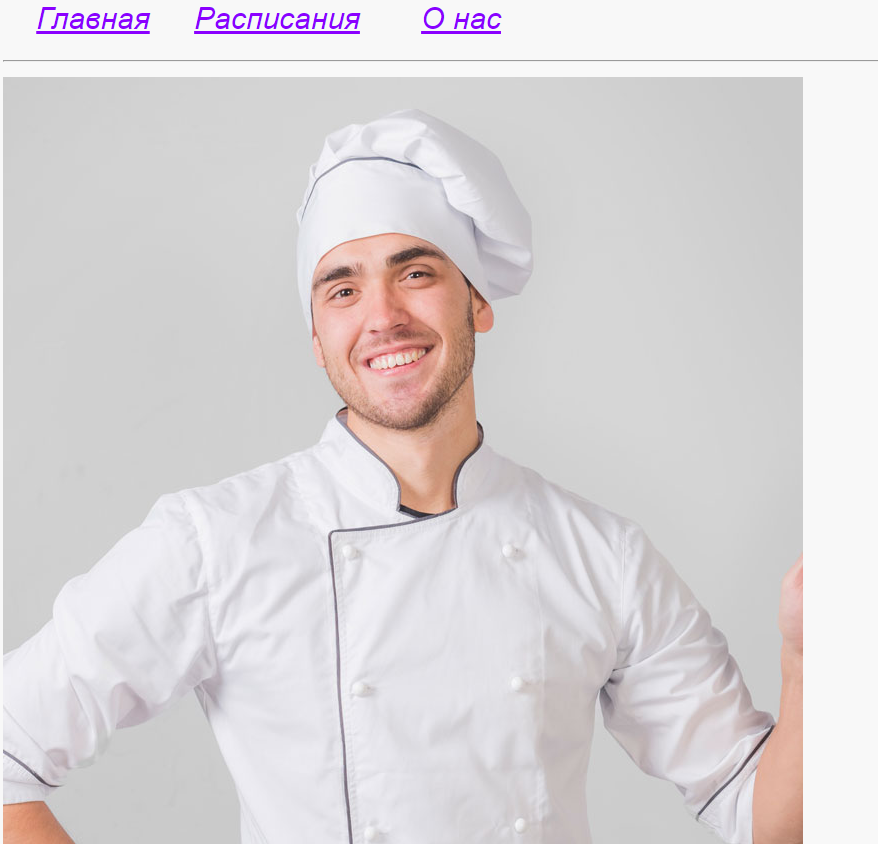
Теперь давайте в качестве контента главной страницы **index.html** разместим какое-нибудь изображение:

**Листинг 10. step\_8/templates/index.html**

{% extends "base.html" %}  
{% block title %}  
Главная страница  
{% endblock %}  
{% block menu %}  
 {% include "inc-menu.html" %}  
{% endblock %}  
{% block body %}  
 <**p**><**img src="{{ static }}images/chef-3.jpg" alt="какой-то текст"**></**p**>  
{% endblock %}

Конечно не забываем разместить файл с картинкой в папке ‘staticfiles/images’.

Запустим проект и проверим его работу:



И картинка есть, и статика не потерялась.

# Вид сверху на рукопожатие над деловыми документамиЗаключение

Мы научили наш фреймворк еще одному важному умению – работать со статикой. При этом мы доработали код самого фреймворка, а также функции, обеспечивающей шаблонизацию. Кроме того, проект пополнился рядом служебных файлов, например, **settings.py** и **content\_types.py**. Мы последовательно разобрали все сложные моменты кода.

* Подключение статики – нетривиальная задача. Для этого необходимо настроить работу фреймворка так, чтобы он при рендеринге страниц обеспечивал и возврат содержимого статических файлов в байтовом формате.
* Также необходимо настроить шаблонизатор, чтобы при работе с адресами наподобие ‘/static/’ они воспринимались именно как подключение статических файлов, т.е. js-скриптов, css-файлов, изображений.
* Также, как и в случае с Django, нам необходимо настроить специальный файл **settings.py**, где прописать пути, которые будут восприниматься фреймворком, как доступ к статическим файлам.

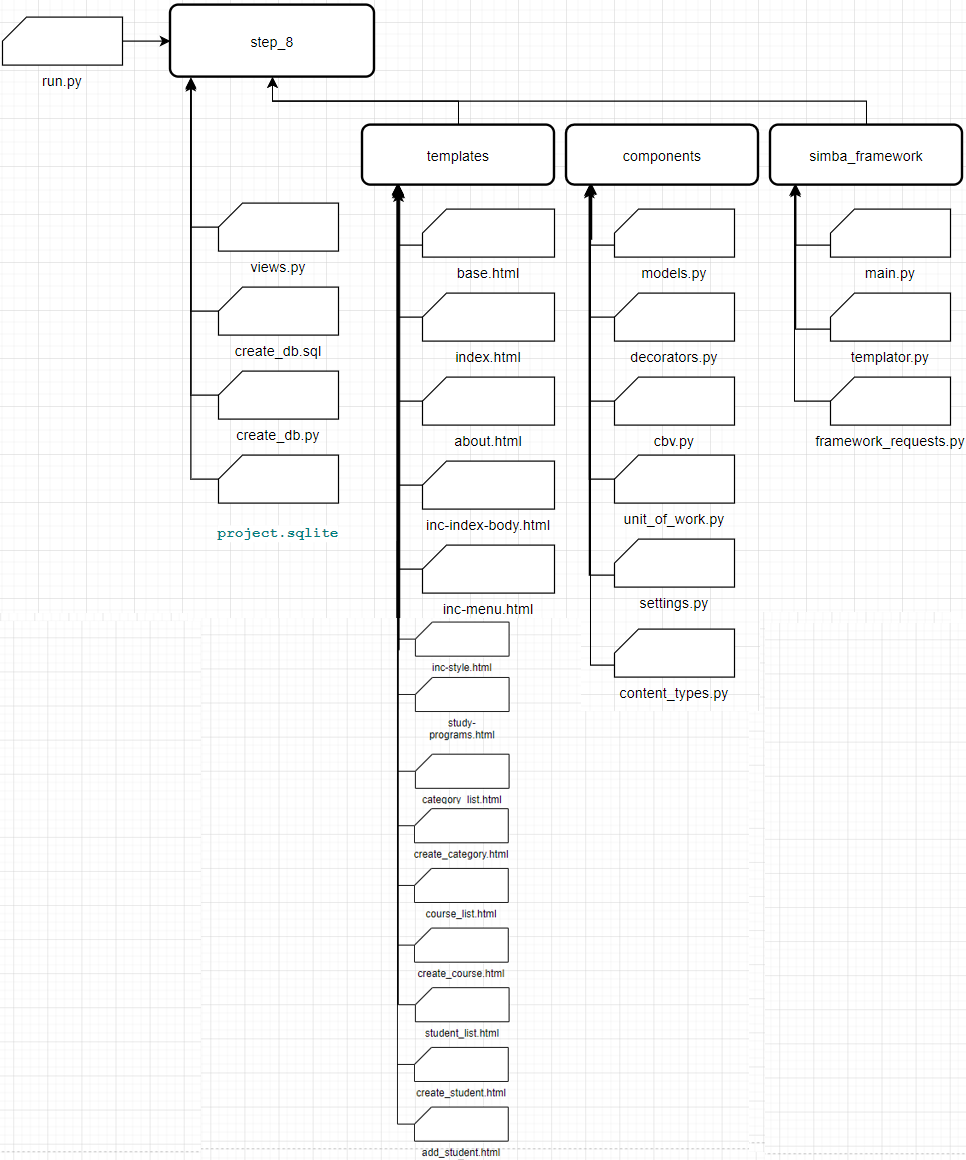
Текущая структура предложенного WSGI-фреймворка и создаваемого на его основе веб-приложения приведена на рис. 1.

Рис. 1. Структура проекта (фреймворк + веб-приложение)