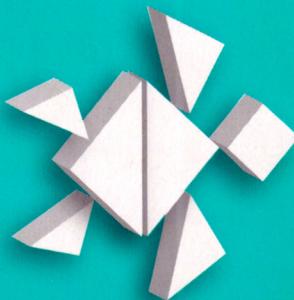
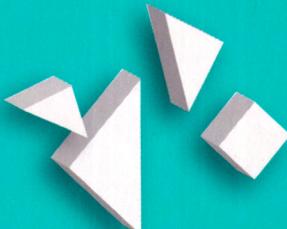
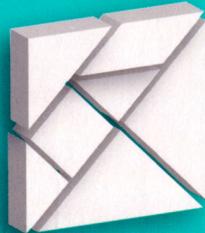


Анатолий Постолит

# Python, Django и Bootstrap

для начинающих



Материалы  
на [www.bhv.ru](http://www.bhv.ru)  
.ru

bhv®

Анатолий Постолит

# Python, Django и Bootstrap

для начинающих

Санкт-Петербург

«БХВ-Петербург»

2023

УДК 004.43  
ББК 32.973.26-018.1  
П63

**Постолит А. В.**

- П63 Python, Django и Bootstrap для начинающих. — СПб.: БХВ-Петербург, 2023. — 624 с.: ил. — (Для начинающих)  
ISBN 978-5-9775-1807-9

Книга посвящена вопросам разработки веб-приложений с использованием языка Python, фреймворков Django, Bootstrap и интерактивной среды разработки PyCharm. Рассмотрены основные технологии и рабочие инструменты создания веб-приложений. Описаны фреймворки Django, Bootstrap и структура создаваемых веб-приложений. На простых примерах показана обработка и маршрутизация запросов пользователей, формирование ответных веб-страниц. Рассмотрено создание шаблонов веб-страниц и форм для пользователей. Показано взаимодействие пользователей с различными типами баз данных через модели. Описана работа с базами данных через встроенные в Django классы без использования SQL-запросов. Приведен пошаговый пример создания сайта — от его проектирования до формирования программных модулей и развертывания сайта в Интернете с базами данных SQLite и MySQL. Электронный архив на сайте издательства содержит коды всех примеров.

*Для программистов*

УДК 004.43  
ББК 32.973.26-018.1

**Группа подготовки издания:**

Руководитель проекта	<i>Евгений Рыбаков</i>
Зав. редакцией	<i>Людмила Гауль</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн серии	<i>Мариной Дамбировой</i>
Оформление обложки	<i>Зои Канторович</i>

Подписано в печать 05.06.23.  
Формат 70×100<sup>1</sup>/<sub>16</sub>. Печать офсетная. Усл. печ. л. 50,31.  
Тираж 2000 экз. Заказ № 6932.  
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.  
Отпечатано с готового оригинал-макета  
ООО "Принт-М", 142300, М.О., г. Чехов, ул. Полиграфистов, д. 1

ISBN 978-5-9775-1807-9

© ООО "БХВ", 2023  
© Оформление. ООО "БХВ-Петербург", 2023

# Оглавление

[https://t.me/it\\_boooks/2](https://t.me/it_boooks/2)

<b>Предисловие .....</b>	<b>9</b>
<b>Глава 1. Инструментальные средства для разработки веб-приложений .....</b>	<b>15</b>
1.1. Интерпретатор Python .....	16
1.1.1. Установка Python в Windows.....	17
1.1.2. Установка Python в Linux .....	20
1.1.3. Проверка интерпретатора Python.....	20
1.2. Интерактивная среда разработки программного кода PyCharm.....	21
1.2.1. Установка PyCharm в Windows .....	22
1.2.2. Установка PyCharm в Linux .....	24
1.2.3. Проверка PyCharm .....	25
1.3. Установка пакетов в Python с использованием менеджера пакетов pip .....	28
1.3.1. Репозиторий пакетов программных средств PyPI.....	28
1.3.2. pip — менеджер пакетов в Python.....	29
1.3.3. Использование менеджера пакетов pip .....	30
1.4. Фреймворк Django для разработки веб-приложений.....	31
1.5. Менеджер баз данных SQLiteStudio.....	35
1.6. Краткие итоги.....	37
<b>Глава 2. Веб-технологии и базовые сведения об HTML.....</b>	<b>39</b>
2.1. Базовые сведения о веб-технологиях .....	39
2.1.1. Технологии клиентского программирования .....	41
2.1.2. Технологии серверного программирования .....	42
2.1.3. Фреймворки Django и Bootstrap для разработки веб-приложений .....	43
2.2. Базовые сведения о HTML.....	45
2.2.1. Теги для представления текста на HTML-страницах.....	47
2.2.2. Списки.....	49
2.2.3. Таблицы .....	51
2.2.4. Тег <i>div</i> .....	55
2.2.5. Гиперссылки .....	56
2.3. Каскадные таблицы стилей (CSS) .....	56
2.4. Возможности использования JavaScript .....	58
2.5. Краткие итоги.....	60

<b>Глава 3. Макетирование HTML-страниц с фреймворком Bootstrap.....</b>	<b>61</b>
3.1. Технологические возможности фреймворка Bootstrap .....	61
3.2. Получение файлов фреймворка Bootstrap .....	63
3.3. Контейнеры и сетка Bootstrap.....	66
3.3.1. Адаптивные контейнеры .....	67
3.3.2. Ряды или строки ( <i>row</i> ) .....	69
3.3.3. Адаптивные блоки ( <i>col</i> ) .....	69
3.3.4. Адаптивные блоки без указания числа колонок .....	71
3.3.5. Расположение адаптивных блоков .....	71
3.4. Верстка макета HTML-страниц.....	72
3.5. Подключение файлов фреймворка Bootstrap к проекту .....	75
3.6. Задание цвета элементам HTML-страниц .....	77
3.7. Задание отступов элементам макета HTML-страниц .....	80
3.8. Выравнивание содержимого в адаптивных блоках HTML-страниц .....	84
3.9. Обозначение границ элементов макета HTML-страниц .....	86
3.10. Пример использования адаптивных контейнеров .....	93
3.11. Таблицы Bootstrap .....	97
3.12. Краткие итоги.....	102
<b>Глава 4. Знакомимся с фреймворком Django .....</b>	<b>103</b>
4.1. Общие представления о Django.....	103
4.2. Структура приложений на Django .....	105
4.3. Первый проект на Django.....	107
4.4. Первое приложение на Django.....	115
4.5. Краткие итоги.....	121
<b>Глава 5. Представления и маршрутизация.....</b>	<b>123</b>
5.1. Обработка запросов пользователей.....	123
5.2. Маршрутизация запросов пользователей в функциях <i>path</i> и <i>re_path</i> .....	126
5.3. Очередность маршрутов .....	129
5.4. Основные элементы синтаксиса регулярных выражений .....	129
5.5. Параметры представлений .....	130
5.5.1. Определение параметров через функцию <i>re_path()</i> .....	130
5.5.2. Определение параметров через функцию <i>path()</i> .....	134
5.5.3. Определение параметров по умолчанию в функции <i>path()</i> .....	136
5.6. Параметры строки запроса пользователя .....	137
5.7. Переадресация и отправка пользователю статусных кодов .....	140
5.7.1. Переадресация .....	140
5.7.2. Отправка пользователю статусных кодов .....	142
5.8. Краткие итоги.....	144
<b>Глава 6. Шаблоны в Django .....</b>	<b>145</b>
6.1. Создание простейшего шаблона.....	145
6.2. Создание каталога для шаблонов приложений .....	152
6.3. Класс <i>TemplateResponse</i> .....	155
6.4. Язык шаблонов (DTL) .....	156
6.5. Передача данных в шаблоны через переменные .....	158
6.6. Передача в шаблон сложных данных .....	161
6.7. Использование тегов в шаблонах Django .....	163

6.8. Статические файлы в шаблонах Django.....	172
6.8.1. Основы каскадных таблиц стилей .....	172
6.8.2. Использование статических файлов в шаблонах Django .....	176
6.9. Использование класса <i>TemplateView</i> для вызова шаблонов HTML-страниц .....	182
6.10. Наследование шаблонов.....	187
6.11. Создание многостраничного сайта на основе шаблонов Django .....	190
6.12. Формирование URL-адресов в шаблонах Django .....	197
6.13. Интеграция шаблонов Django с фреймворком Bootstrap .....	200
6.14. Использование специальных тегов в шаблонах Django .....	203
6.14.1. Тег для вывода текущей даты и времени .....	203
6.14.2. Теги Bootstrap для вывода информации в адаптивных блоках .....	205
6.14.3. Теги Bootstrap и Django для представления списков в виде таблицы .....	207
6.15. Краткие итоги.....	209
<b>Глава 7. Формы .....</b>	<b>211</b>
7.1. Процесс управления формами в Django .....	211
7.2. Определение форм.....	214
7.3. Использование полей в формах Django .....	220
7.3.1. Настройка среды для изучения полей разных типов.....	220
7.3.2. Типы полей в формах Django и их общие параметры .....	222
7.3.3. Поле <i>BooleanField</i> для выбора решения: да/нет .....	225
7.3.4. Поле <i>CharField</i> для ввода текста .....	226
7.3.5. Поле <i>ChoiceField</i> для выбора данных из списка .....	228
7.3.6. Поле <i>DateField</i> для ввода даты .....	229
7.3.7. Поле <i>DateTimeField</i> для ввода даты и времени .....	230
7.3.8. Поле <i>DecimalField</i> для ввода десятичных чисел .....	230
7.3.9. Поле <i>DurationField</i> для ввода промежутка времени .....	232
7.3.10. Поле <i>EmailField</i> для ввода электронного адреса.....	233
7.3.11. Поле <i>FileField</i> для выбора файлов.....	234
7.3.12. Поле <i>FilePathField</i> для создания списка файлов .....	235
7.3.13. Поле <i>FloatField</i> для ввода чисел с плавающей точкой .....	238
7.3.14. Поле <i>GenericIPAddressField</i> для ввода IP-адреса .....	238
7.3.15. Поле <i>ImageField</i> для выбора файлов изображений .....	239
7.3.16. Поле <i>IntegerField</i> для ввода целых чисел.....	240
7.3.17. Поле <i>JsonField</i> для данных формата JSON .....	241
7.3.18. Поле <i>MultipleChoiceField</i> для выбора данных из списка .....	243
7.3.19. Поле <i>NullBooleanField</i> для выбора решения: да/нет .....	244
7.3.20. Поле <i>RegexField</i> для ввода текста .....	245
7.3.21. Поле <i>SlugField</i> для ввода текста .....	246
7.3.22. Поле <i>TimeField</i> для ввода времени .....	246
7.3.23. Поле <i>TypedChoiceField</i> для выбора данных из списка.....	247
7.3.24. Поле <i>TypedMultipleChoiceField</i> для выбора данных из списка .....	248
7.3.25. Поле <i>URLField</i> для ввода универсального указателя ресурса (URL) .....	250
7.3.26. Поле <i>UUIDField</i> для ввода универсального уникального идентификатора UUID .....	251
7.4. Встроенные классы для создания сложных полей.....	252
7.4.1. Поле <i>ComboField</i> для ввода текста с проверкой соответствия заданным форматам.....	252
7.4.2. Поле <i>MultiValueField</i> для создания сложных компоновок из нескольких полей .....	253
7.4.3. Поле <i>SplitDateTimeField</i> для раздельного ввода даты и времени .....	254

7.5. Настройка формы и ее полей.....	255
7.5.1. Изменение внешнего вида поля с помощью параметра <i>widget</i> .....	255
7.5.2. Задание начальных значений полей с помощью свойства <i>initial</i> .....	257
7.5.3. Задание порядка следования полей на форме.....	258
7.5.4. Задание подсказок к полям формы.....	259
7.5.5. Настройки вида формы.....	260
7.5.6. Проверка (валидация) данных.....	262
7.5.7. Детальная настройка полей формы .....	267
7.5.8. Присвоение стилей полям формы.....	270
7.6. Использование в формах POST-запросов для отправки данных на сервер.....	275
7.7. Краткие итоги.....	278
<b>Глава 8. Модели данных Django.....</b>	<b>279</b>
8.1. Создание моделей и миграции базы данных .....	280
8.2. Типы полей в модели данных Django .....	284
8.3. Манипуляция с данными в Django на основе CRUD.....	287
8.3.1. Добавление данных в БД.....	287
8.3.2. Чтение данных из БД .....	288
Метод <i>get()</i> .....	288
Метод <i>get_or_create()</i> .....	288
Метод <i>all()</i> .....	289
Метод <i>count()</i> .....	289
Метод <i>filter()</i> .....	289
Метод <i>exclude()</i> .....	289
Метод <i>in_bulk()</i> .....	289
8.3.3. Обновление данных в БД .....	290
8.3.4. Удаление данных из БД .....	291
8.3.5. Просмотр строки SQL-запроса к базе данных.....	291
8.4. Общие принципы взаимодействия форм с моделями данных и шаблонами Django .....	292
8.4.1. Создание форм на основе классов <i>Form</i> и <i>ModelForm</i> .....	293
8.4.2. Связывание форм с представлениями ( <i>view</i> ) .....	294
8.4.3. Связывание представлений ( <i>view</i> ) с шаблонами форм .....	295
8.5. Организация связей между таблицами в БД через модели данных.....	296
8.5.1. Организация связей между таблицами «один ко многим» .....	296
8.5.2. Организация связей между таблицами «многие ко многим» .....	301
8.5.3. Организация связей между таблицами «один к одному».....	305
8.6. Пример работы с объектами модели данных (чтение и запись информации в БД).....	308
8.7. Пример работы с объектами модели данных: редактирование и удаление информации из БД.....	315
8.8. Работа с изображениями и файлами в формах Django .....	321
8.8.1. Загрузка изображений.....	321
8.8.2. Загрузка и отображение файлов PDF в формах Django .....	332
8.8.3. Загрузка и отображение видеофайлов в формах Django.....	339
8.8.4. Загрузка и озвучивание аудиофайлов в формах Django .....	348
8.9. Краткие итоги.....	356
<b>Глава 9. Пример создания веб-сайта на Django .....</b>	<b>357</b>
9.1. Создание структуры сайта при помощи Django.....	357
9.2. Установка дополнительных пакетов и настройка параметров сайта «Мир книг» .....	366
9.3. Разработка структуры моделей данных сайта «Мир книг».....	370

9.4. Основные элементы моделей данных в Django .....	373
9.4.1. Поля и их аргументы в моделях данных .....	373
9.4.2. Метаданные в моделях Django .....	376
9.4.3. Методы в моделях Django .....	377
9.4.4. Методы работы с данными в моделях Django .....	377
9.5. Формирование моделей данных для сайта «Мир книг» .....	379
9.5.1. Модель для хранения жанров книг .....	380
9.5.2. Модель для хранения языков книг .....	381
9.5.3. Модель для хранения наименования издательства .....	381
9.5.4. Модель для хранения авторов книг .....	382
9.5.5. Модель для хранения книг .....	382
9.5.6. Модель для хранения отдельных экземпляров книг и их статуса.....	387
9.6. Административная панель Django Admin.....	391
9.6.1. Регистрация моделей данных в Django Admin .....	391
9.6.2. Работа с данными в Django Admin.....	392
9.7. Изменение конфигурации административной панели Django .....	402
9.7.1. Регистрация класса <i>ModelAdmin</i> .....	403
9.7.2. Настройка отображения списков .....	404
9.7.3. Добавление фильтров к спискам.....	407
9.7.4. Формирование макета с подробным представлением элемента списка.....	409
9.7.5. Разделение страницы на секции с отображением связанной информации .....	411
9.7.6. Встроенное редактирование связанных записей .....	412
9.8. Работа с файлами и изображениями в административной панели Django .....	416
9.9. Краткие итоги.....	422

## Глава 10. Пример создания веб-интерфейса для пользователей сайта

«Мир книг».....	423
10.1. Последовательность создания пользовательских страниц сайта «Мир книг».....	423
10.2. Определение перечня и URL-адресов страниц сайта «Мир книг» .....	424
10.3. Создание главной страницы сайта «Мир книг» .....	425
10.3.1. Создание URL-преобразования.....	425
10.3.2. Создание упрощенного представления ( <i>view</i> ) .....	427
10.3.3. Изменение представления ( <i>view</i> ) главной страницы сайта.....	432
10.3.4. Модификация шаблона главной страницы сайта «Мир книг» .....	433
10.4. Создание страницы со списком книг на основе класса <i>ListView</i> .....	438
10.5. Создание страницы с детальной информацией о книге на основе класса <i>DetailView</i> .....	443
10.6. Постраничный вывод большого числа записей из БД (класс <i>Paginator</i> ).....	448
10.7. Создание страницы со списком авторов на основе класса <i>ListView</i> .....	453
10.8. Создание страницы с детальной информацией об авторе книги на основе класса <i>DetailView</i> .....	456
10.9. Создание страниц О компании и Контакты .....	460
10.10. Краткие итоги.....	468

## Глава 11. Расширение возможностей администрирования сайта

«Мир книг» и создание пользовательских форм .....	469
11.1. Сессии в Django .....	470
11.2. Аутентификация и авторизация пользователей в Django .....	474
11.2.1. Немного об аутентификации пользователей в Django .....	474
11.2.2. Создание отдельных пользователей и групп пользователей .....	475

11.2.3. Создание страницы регистрации пользователя при входе на сайт .....	481
11.2.4. Создание страницы для сброса пароля пользователя .....	486
11.3. Настройка почты для отправки сообщения о смене пароля на реальный электронный адрес .....	494
11.4. Проверка подлинности входа пользователя в систему.....	499
11.5. Формирование страниц сайта для создания заказов на книги.....	502
11.6. Работа с формами .....	512
11.6.1. Краткий обзор форм в Django .....	512
11.6.2. Управление формами в Django .....	514
11.6.3. Форма для ввода и обновления информации об авторах книг на основе класса <i>Form()</i> .....	515
11.6.4. Форма для обновления информации об авторах книг на основе класса <i>ModelForm()</i> .....	527
11.6.5. Форма для ввода и обновления информации о книгах на основе класса <i>ModelForm()</i> .....	533
11.7. Краткие итоги.....	545
<b>Глава 12. Публикация сайта в сети Интернет.....</b>	<b>547</b>
12.1. Подготовка инфраструктуры сайта перед публикацией в сети Интернет .....	547
12.1.1. Окружение развертывания сайта в сети Интернет .....	548
12.1.2. Выбор хостинг-провайдера .....	549
12.2. Подготовка веб-сайта к публикации .....	550
12.3. Размещение веб-сайта на хостинге timeweb .....	552
12.3.1. Регистрация аккаунта пользователя .....	552
12.3.2. Административная панель хостинга timeweb .....	555
12.3.3. Создание на сервере папки для нового сайта .....	559
12.3.4. Создание на сервере виртуального окружения и приложения Django для нового сайта .....	567
12.3.5. Перенос сайта с рабочего компьютера на удаленный сервер .....	574
12.3.6. Смена временного доменного имени на постоянное .....	579
12.4. Краткие итоги.....	581
<b>Глава 13. Приложения Django и MySQL .....</b>	<b>583</b>
13.1. Подготовка инфраструктуры сайта для перехода на MySQL .....	583
13.2. Инсталляция сервера MySQL .....	584
13.3. Создание базы данных .....	584
13.4. Создание проекта Django с базой данных MySQL на локальном компьютере .....	588
13.5. Создание инфраструктуры на удаленном сервере для сайта с базой данных на MySQL .....	591
13.6. Создание базы данных MySQL на удаленном сервере .....	599
13.7. Перенос сайта с локального компьютера на публичный сервер .....	603
13.8. Краткие итоги.....	611
<b>Послесловие.....</b>	<b>613</b>
<b>Список источников и литературы.....</b>	<b>614</b>
<b>Приложение. Описание электронного архива.....</b>	<b>617</b>
<b>Предметный указатель .....</b>	<b>618</b>

# Предисловие

С развитием цифровых технологий и уходом в прошлое ряда популярных ранее специальностей молодые люди все чаще встают перед выбором перспективной, интересной и актуальной профессии. Международное интернет-сообщество считает, что одним из самых перспективных вариантов такого выбора является профессия веб-программиста. Именно эти специалисты формируют облик сети Интернет и создают технологические тренды будущего.

Каждую секунду в Интернете появляется от 3 до 5 новых сайтов, а каждую минуту — 80 новых интернет-пользователей. Все это технологическое «цунами» управляет разумом и умелыми руками веб-разработчиков. Зарплата этих специалистов вполне соответствует важности и актуальности их деятельности. Даже начинающие программисты на отечественном рынке могут рассчитывать на заработную плату от 50 тыс. рублей в месяц, а опытные программисты в России и за рубежом имеют доход, превышающий указанную цифру в десятки раз.

Специалисты ИТ-технологий уже много лет подряд занимают первые позиции в рейтингах кадровых агентств, как представители наиболее востребованных профессий на отечественном и мировом рынке труда. Постоянная нехватка квалифицированных программистов дает высокий шанс молодым и целеустремленным новичкам для входа в эту профессию.

Согласно данным Ассоциации предприятий компьютерных и информационных технологий (АПКИТ) спрос на ИТ-специалистов ежегодно остается на высоком уровне, однако кандидатов по-прежнему не хватает. В ближайшие несколько лет дефицит будет только нарастать, т. к. ИТ-специалисты нужны повсеместно. Если раньше ими интересовались сугубо профильные компании (разработчики софта и онлайн-платформ), то сейчас квалифицированные кадры ИТ-профилей требуются практически везде. И речь здесь идет не о дефиците, а о катастрофическом дефиците. Уже много лет подряд сервис по поиску работы SuperJob фиксирует огромный неудовлетворенный спрос на ИТ-специалистов.

Рассматриваемый в этой книге язык программирования Python — один из самых популярных языков программирования, и области его применения только расширяются. Последние несколько лет он входит в Топ-3 самых востребованных языков, а в 2022 году вышел на первое место. Задействуя Python, можно решать множество научных проблем и задач в области бизнеса. К нему обращаются ученые (математики, физики, биологи),

т. к. изучить его не слишком сложно. Он используется при реализации нейронных сетей, для написания веб-сайтов и мобильных приложений. В целом это универсальный язык, входящий в тройку языков для анализа больших данных. В течение последних 5 лет Python-разработчики востребованы на рынке труда, специалистов в этой сфере до сих пор не хватает.

Еще одна причина, по которой следует обратить внимание на использование языка Python, фреймворков Django и Bootstrap для веб-разработки, — это развитие систем искусственного интеллекта. На Python реализовано большое число библиотек, облегчающих создание программного обеспечения для систем искусственного интеллекта, Django обеспечивает разработку серверной части, а Bootstrap — клиентской части веб-приложений с развертыванием их в сети Интернет. С помощью этого инструментария можно создавать приложения для беспилотных автомобилей, для интеллектуальных транспортных систем, для телемедицины, систем обучения, распознавания объектов в системах безопасности и т. п. Это очень востребованные и актуальные сферы, где наблюдается еще большая потребность в специалистах. Здесь талантливые и целеустремленные молодые люди могут найти как возможность самореализации, так и достойную оплату своего труда.

Каждая организация, принимающая на работу IT-специалиста, требует знаний, которые будут полезны именно в ее работе. Однако общее правило таково, что чем больше популярных и необходимых языков программирования, фреймворков и баз данных вы знаете (Js, HTML, C#, C++, Python, PHP, Django, SQL) и чем больше ваш опыт работы, тем выше шансы на удачное трудоустройство и достойную зарплату.

Перед теми, кто решил освоить специальность веб-программиста, встает непростой выбор — с чего же правильно начать. Конечно, всегда существует возможность получить полноценное IT-образование в одном из ведущих технических вузов ранга МГУ, МГТУ им. Н. Баумана, СПбГУ, МФТИ, ИТМО. Но обучение в них обойдется в круглую сумму от 60 до 350 тыс. рублей в год. Существует и более быстрый и дешевый вариант стать веб-разработчиком «с нуля», пройдя краткосрочные онлайн-курсы. Однако практикующие программисты уверяют, что на начальном этапе самый правильный способ обучиться веб-программированию — освоить его самостоятельно. Так можно не только избежать серьезных расходов, но и получить только те практические навыки, которые пригодятся в будущей работе. Полученные самостоятельно базовые знания впоследствии можно расширить, обучаясь уже на соответствующих курсах или в специализированном вузе.

Эта книга предназначена как для начинающих программистов (школьников и студентов), так и для специалистов с опытом, которые планируют разрабатывать или уже занимаются разработкой веб-приложений с использованием Python. Сразу следует отметить, что веб-программирование требует от разработчиков больших знаний, умений и усилий, чем программирование традиционных приложений. Здесь, кроме основного языка, который реализует логику приложения, необходимо еще и знание структуры HTML-документов, основ работы с базами данных, принципов взаимодействия удаленных пользователей с веб-приложением. Если некоторые разделы книги вам покажутся трудными для понимания и восприятия, то не стоит отчаиваться. Нужно просто последовательно, по шагам повторить приведенные в книге примеры. А когда вы увидите результаты работы программного кода, появится ясность — как работает тот или иной элемент изучаемого фреймворка. Конечно, для тех, кто собирается заниматься создани-

ем веб-приложений на основе Django, необходимы базовые знания Python: переменные, списки, кортежи, словари, функции, классы. Эти знания можно получить из специальной литературы.

В книге рассмотрены практически все элементарные действия, которые выполняют программисты, работая над реализацией веб-приложений, приведено множество примеров и проверенных программных модулей. Рассмотрены базовые классы фреймворка Django, методы и свойства каждого из классов и примеры их использования. Книга, как уже отмечалось, предназначена как для начинающих программистов, приступивших к освоению языка Python, так и специалистов, имеющих опыт программирования на других языках. В этом издании в меньшей степени освещены вопросы дизайна веб-приложений, а больше внимания уделено технологиям разработки веб-приложений на практических примерах. Тем не менее в книге приводятся базовые сведения о фреймворке Bootstrap и его использовании вместе с Django для макетирования и разработки дизайна HTML-страниц.

*Глава 1* книги посвящена инструментальным средствам, которые применяются для разработки веб-приложений. В ней сказано, как установить на локальный компьютер Python, среду для написания программного кода PyCharm, фреймворк Django, менеджер для работы с базами данных SQLiteStudio. Кроме того, показано, как установить различные дополнительные пакеты в Python с использованием менеджера пакетов pip.

*Глава 2* книги посвящена знакомству с веб-технологиями, рассмотрены принципиальные различия между приложениями, работающими на сервере и на стороне клиента. Показаны различия между такими понятиями, как *фронтенд-разработка* и *бэкенд-разработка*. Приведены базовые сведения о HTML-страницах, их структуре и основных тегах языка HTML, позволяющих выводить и форматировать информацию. Представлен перечень и краткое описание языков программирования, позволяющих создавать веб-приложения, даны базовые сведения о каскадных таблицах стилей (CSS), и о JavaScript.

В *главе 3* рассматриваются возможности макетирования HTML-страниц с помощью фреймворка Bootstrap. Из этой главы вы узнаете, как можно получить файлы фреймворка Bootstrap и подключить их к своему проекту, что такое контейнеры и сетка Bootstrap. На простых примерах показано, как можно разбить HTML-страницы на адаптивные блоки и придать им привлекательный внешний вид. Это очень полезный и достаточно популярный инструмент, который позволяет выполнить оформление внешнего вида сайта.

*Глава 4* посвящена основным понятиям и определениям, которые приняты во фреймворке Django. В ней также описана структура приложений на Django. С использованием интерактивной среды PyCharm мы создадим здесь первый простейший проект на Django и сформируем в этом проекте приложение.

В *главе 5* приведены основные понятия о представлениях (view) и маршрутизации запросов пользователей. Это весьма важные компоненты, которые определяют, что должно делать веб-приложение при поступлении различных запросов от удаленных пользователей. Здесь же описан синтаксис так называемых *регулярных выражений*, которые преобразуют запросы пользователей в адреса перехода к тем или иным страницам сайта.

В главе 6 рассмотрены вопросы создания шаблонов HTML-страниц. На конкретных примерах показано, как передать в шаблоны простые и сложные данные. Приведены примеры использования статичных файлов в приложениях на Django. Рассмотрена возможность расширения шаблонов HTML-страниц на основе базового шаблона — это, по сути, основной прием, который применяется практически на всех сайтах. Приведены также примеры использования в шаблонах HTML-страниц специальных тегов (для вывода текущей даты и времени, вывода информации по условию, вывода информации в цикле и для задания значений переменным).

В главе 7 сделан обзор пользовательских форм. Формы — это основной интерфейс, благодаря которому пользователи имеют возможность вносить информацию в удаленную базу данных. Из этой главы вы узнаете, как использовать в формах POST-запросы, в ней также подробно описаны поля, которые можно задействовать для ввода данных, и приводятся короткие примеры применения каждого типа поля. Кроме того, рассмотрены примеры изменения внешнего вида полей с помощью виджетов (widget), настроек вида полей, валидации данных и стилизации полей на формах Django.

Глава 8 посвящена изучению моделей Django. Модели, по своей сути, представляют собой описание таблиц и полей базы данных (БД), используемой веб-приложением. На основе моделей будут автоматически созданы классы, через свойства и методы которых приложение станет взаимодействовать с базой данных. Соответственно через классы будут выполняться все процедуры работы с данными (добавление, модификация, удаление записей из таблиц БД), при этом отпадает необходимость в написании SQL-запросов — Django будет создавать их самостоятельно и задействовать при манипулировании данными. Кроме того, посредством процедуры миграции Django в автоматическом режиме создаст необходимые таблицы в различных СУБД (SQLite, PostgreSQL, MySQL, Oracle).

В главе 9 мы создадим модели для достаточно простого «учебного» сайта «Мир книг» и соответствующие таблицы в системе управления базами данных (СУБД) SQLite. Возможность работы с данными этого сайта здесь будет показана через встроенную в Django административную панель.

В главе 10 приведен пример создания веб-интерфейса для пользователей сайта «Мир книг». Здесь будет определен перечень страниц сайта и их интернет-адреса (URL), созданы представления (views) для обработки запросов пользователей, базовый шаблон сайта и шаблоны прочих страниц сайта. Это простейшие страницы для получения информации из БД на основе запросов пользователей. Здесь к проекту Django будут подключены файлы Bootstrap и продемонстрировано их взаимодействие.

В главе 11 показано, как можно расширить возможности администрирования сайта «Мир книг» и обеспечить ввод и редактирование данных со стороны удаленных пользователей через формы Django. Здесь рассмотрено такое понятие, как *сессия*, представлены процедуры создания страниц авторизация пользователей, проверки подлинности входа пользователей в систему, изменение внешнего вида страниц для зарегистрированных пользователей. Подробно на примерах показано применение классов `Form` и `ModelForm`.

Глава 12 посвящена теме публикации сайтов, разработанных на Django, в сети Интернет. Эта, казалось бы, несложная процедура требует от разработчика определенной квалификации и навыков. На удаленном сервере потребуется создать соответствующее

окружение для Python, подгрузить необходимые библиотеки и модули, подключить нужную СУБД, настроить пути к статичным файлам рисунков и стилей. В качестве примера мы выполним процедуру публикации «учебного» сайта «Мир книг» на российском сетевом ресурсе timeweb с использованием СУБД SQLite. В этой же главе вы найдете информацию о том, как зарегистрировать доменное имя и подключить его к сайту, опубликованному на публичном сервере.

В главе 13 показано, как можно переключить разработанное приложение с СУБД SQLite на MySQL. Из этой главы вы узнаете, как установить сервер MySQL на локальный компьютер, создать базу данных на MySQL и подключить к ней проект Django. Также будет рассказано о том, как можно создать инфраструктуру на удаленном сервере для развертывания сайта Django с СУБД MySQL и перенести свой сайт с локального компьютера на публичный сервер.

На протяжении всей книги раскрываемые вопросы сопровождаются достаточно упрощенными, но полностью законченными примерами. Ход решения той или иной задачи показан на большом количестве иллюстративного материала. Желательно изучение тех или иных разделов осуществлять, сидя непосредственно за компьютером, — тогда вы сможете последовательно повторять выполнение тех шагов, которые описаны в примерах, и тут же видеть результаты своих действий, что в значительной степени облегчит восприятие материала книги. Наилучший способ обучения — это практика. Все листинги программ приведены на языке Python, а шаблоны веб-страниц — в виде HTML-файлов. Это прекрасная возможность познакомиться с языком программирования Python и понять, насколько он прост и удобен в использовании.

### **ЭЛЕКТРОННЫЙ АРХИВ**

Сопровождающий книгу электронный архив содержит программный код ряда наиболее важных листингов книги (см. *приложение*). Архив доступен для скачивания с сервера издательства «БХВ» по ссылке <https://zip.bhv.ru/9785977518079.zip>, ссылка на него также ведет со страницы книги на сайте <https://bhv.ru>. Пример сайта, разработанного в данной книге, вы можете посмотреть в Интернете по адресу: [putbook.ru](http://putbook.ru).

Итак, если вас заинтересовали вопросы создания веб-приложений с использованием Python, Django и Bootstrap в среде PyCharm, то самое время перейти к изучению материалов этой книги.



# ГЛАВА 1



## Инструментальные средства для разработки веб-приложений

[https://t.me/it\\_boooks/2](https://t.me/it_boooks/2)

Для разработки веб-приложений существует множество языков программирования, каждый из которых имеет свои особенности. Следует отметить, что веб-приложения состоят из двух частей — клиентской (frontend) и серверной (backend). Исходя из этого, программистов, которые занимаются веб-приложениями, делят на две категории: frontend-разработчики и backend-разработчики, и они используют разные языки программирования. Для разработки клиентской части веб-приложения применяют язык гипертекстовой разметки HTML в сочетании с CSS и JavaScript. А вот серверную часть можно разрабатывать с помощью совершенно разных языков программирования: PHP, Python, Ruby, C#, Perl. Но из них хочется выделить Python, который стал наиболее распространенным и универсальным средством разработки программного кода на протяжении уже более тридцати лет.

Python — интерпретируемый язык программирования — в конце 1989 года создал Гвидо Ван Россум, и он очень быстро стал популярным и востребованным у программистов. В подтверждение этого можно упомянуть компании-гиганты: Google, Microsoft, Facebook, Yandex и многие другие, которые используют Python для реализации глобальных проектов.

Область применения Python очень обширна — это и обработка научных данных, и системы управления жизнеобеспечением, а также игры, веб-ресурсы, системы искусственного интеллекта. За все время существования Python динамично развивался. Для него создавались стандартные библиотеки, обеспечивающие поддержку современных технологий — например, для работы с базами данных, протоколами Интернета, электронной почтой, машинным обучением и многим другим.

Ускорить процесс написания программного кода позволяет специализированная инструментальная среда — так называемая *интегрированная среда разработки* (IDE, Integrated Development Environment). Эта среда включает полный комплект средств, необходимых для эффективного программирования на Python. Обычно в состав IDE входят текстовый редактор, компилятор или интерпретатор, отладчик и другое программное обеспечение. Применение IDE позволяет увеличить скорость разработки программ (при условии предварительного обучения работе с такой инструментальной средой). На сегодняшний день наиболее популярная среда для разработки приложений на Python — IDE PyCharm. Именно это программное обеспечение будет использовано при написании программного кода для данной книги.

Возможность создавать веб-приложения не встроена в основные функции Python. Для реализации задач подобного класса на Python нужен дополнительный инструментарий. Таким инструментарием является фреймворк Django, который считается лучшим фреймворком, написанным на Python, предназначенным для написания серверной части веб-приложений. Этот инструмент хорошо подходит для создания сайтов, работающих с базами данных, он делает веб-разработку на Python очень качественной и удобной.

Что касается разработки клиентской части веб-приложений, то в настоящее время наиболее популярным инструментом считается фреймворк Bootstrap. С его помощью достаточно просто и удобно создавать пользовательский интерфейс, который адаптируется под экраны разного размера и разрешения. Страницы сайтов, созданные на основе Bootstrap, хорошо читаются и имеют привлекательный вид на больших экранах настольных компьютеров, на средних экранах ноутбуков и планшетов, на малых экранах смартфонов. Вместе с Django можно использовать как оригинальный фреймворк Bootstrap, так и его различные адаптированные версии, например django-bootstrap-v5. В данной книге нам понадобятся лишь некоторые элементы фреймворка Bootstrap, предназначенные для макетирования HTML-страниц.

Большинство современных веб-приложений обеспечивают взаимодействие пользователей с базами данных, в которых хранятся сведения о зарегистрированных пользователях, тексты, изображения, аудио- и видеоконтент. Доступ к содержимому баз данных осуществляется с помощью языка SQL. Несмотря на то что Django позволяет обойти прямое использование SQL через модели данных, знание основ SQL важно для веб-разработки. Приложения на Django могут взаимодействовать с различными базами данных, а по умолчанию принята простейшая система управления базами данных (СУБД) SQLite. В процессе работы над примерами книги необходимо будет заглядывать в содержимое таблиц СУБД, а для этого нужно иметь соответствующее программное средство. Работать с СУБД SQLite мы будем посредством простейшего менеджера баз данных SQLiteStudio.

Из материалов этой главы вы также узнаете, как установить следующие компоненты:

- интерпретатор Python;
- интерактивную среду разработки программного кода PyCharm;
- различные дополнительные пакеты в Python с использованием менеджера пакетов pip;
- фреймворк Django.
- менеджер баз данных SQLiteStudio.

## 1.1. Интерпретатор Python

Язык программирования Python — это весьма мощное инструментальное средство для разработки различных систем. Однако наибольшую ценность представляет даже не столько сам язык программирования, сколько набор подключаемых библиотек, на уровне которых уже реализованы все необходимые процедуры и функции. Разработчику достаточно написать несколько десятков строк программного кода, чтобы подклю-

чить требуемые библиотеки, создать набор необходимых объектов, передать им исходные данные и отобразить итоговые результаты.

Для установки интерпретатора Python на компьютер, прежде всего надо загрузить его дистрибутив. Скачать дистрибутив Python можно с официального сайта, перейдя по ссылке <https://www.python.org/downloads/> (рис. 1.1).

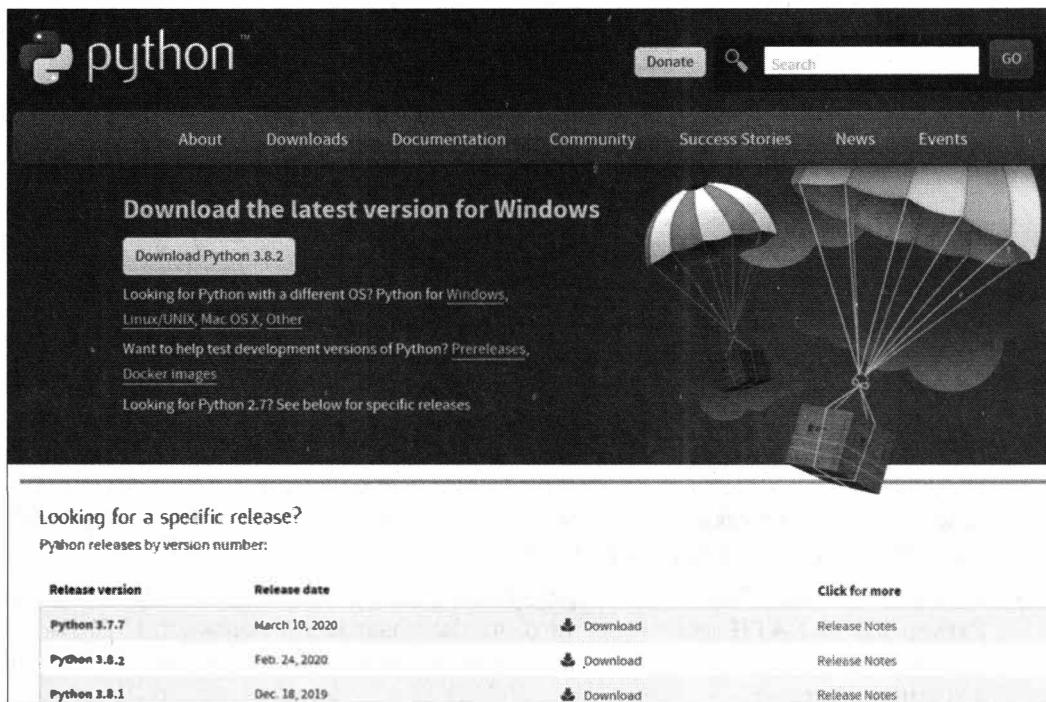


Рис. 1.1. Сайт для скачивания дистрибутива языка программирования Python

На момент написания книги доступен Python версии 3.11, однако здесь мы будем использовать более раннюю версию 3.8, поскольку она стабильно работает на всех версиях Windows, начиная с Windows 7 и выше.

### 1.1.1. Установка Python в Windows

Для операционной системы Windows дистрибутив Python распространяется либо в виде исполняемого файла (с расширением `exe`), либо в виде архивного файла (с расширением `zip`). При написании программного кода для этой книги был использован Python версии 3.8.3.

Порядок установки Python в Windows следующий:

1. Запустите скачанный установочный файл.
2. Выберите способ установки (рис. 1.2).

В открывшемся окне предлагаются два варианта: **Install Now** и **Customize installation**:

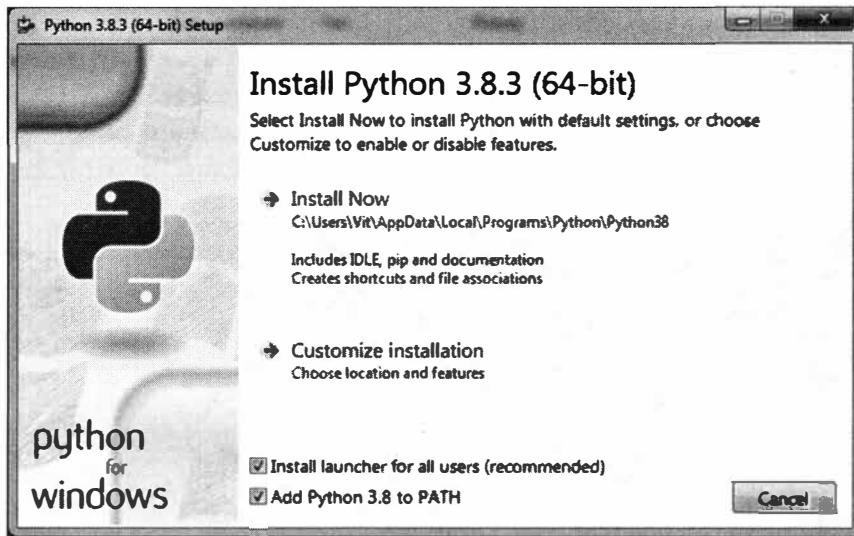


Рис. 1.2. Выбор способа установки Python

- при выборе **Install Now** Python установится в папку по указанному в окне пути. Помимо самого интерпретатора будут инсталлированы IDLE (интегрированная среда разработки), pip (пакетный менеджер) и документация, а также созданы соответствующие ярлыки и установлены связи (ассоциации) файлов, имеющих расширение py, с интерпретатором Python;
  - **Customize installation** — это вариант настраиваемой установки. Опция **Add Python 3.8 to PATH** нужна для того, чтобы появилась возможность запускать интерпретатор без указания полного пути до исполняемого файла при работе в командной строке.
3. Отметьте необходимые опции установки, как показано на рис. 1.3 (доступно при выборе варианта **Customize installation**).

На этом шаге нам предлагается отметить дополнения, устанавливаемые вместе с интерпретатором Python. Рекомендуется выбрать как минимум следующие опции:

- **Documentation** — установка документации;
- **pip** — установка пакетного менеджера pip;
- **tcl/tk and IDLE** — установка интегрированной среды разработки (IDLE) и библиотеки для построения графического интерфейса (tkinter).

4. На следующем шаге в разделе **Advanced Options** (дополнительные опции) выберите место установки, как показано на рис. 1.4 (доступно при выборе варианта **Customize installation**).

Помимо указания пути, этот раздел позволяет внести дополнительные изменения в процесс установки с помощью опций:

- **Install for all users** — установить для всех пользователей. Если не выбрать эту опцию, то будет предложен вариант инсталляции в папку пользователя, устанавливающего интерпретатор;

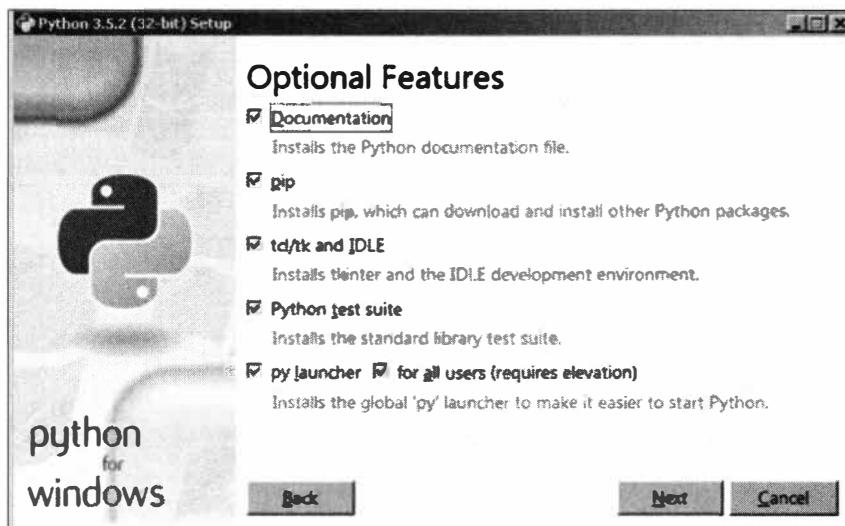


Рис. 1.3. Выбор опций установки Python

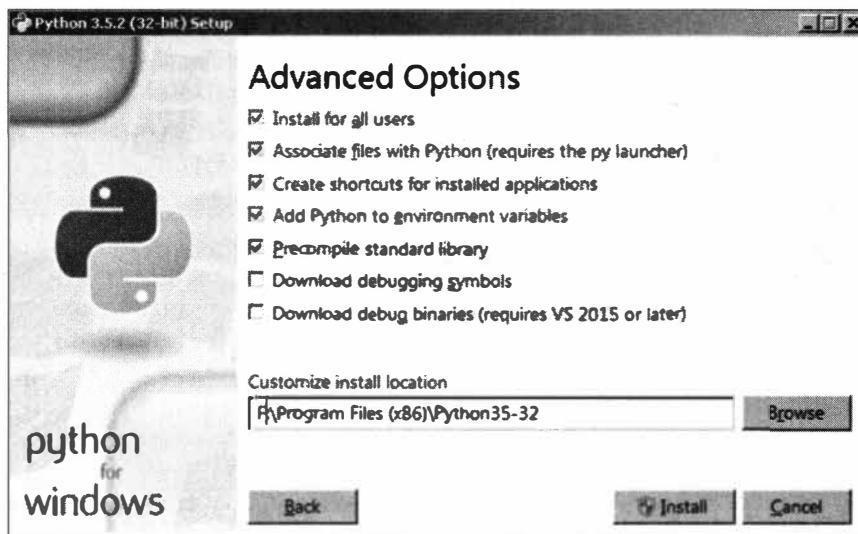


Рис. 1.4. Выбор места установки Python

- **Associate files with Python** — связать файлы, имеющие расширение `py`, с Python. При выборе этой опции будут внесены изменения в Windows, позволяющие Python запускать скрипты по двойному щелчку мыши;
- **Create shortcuts for installed applications** — создать ярлыки для запуска приложений;
- **Add Python to environment variables** — добавить пути до интерпретатора Python в переменную PATH;
- **Precompile standard library** — провести перекомпиляцию стандартной библиотеки.

Последние два пункта (см. рис. 1.4) связаны с загрузкой компонентов для отладки, их мы устанавливать не будем.

5. После установки Python вас ждет следующее сообщение об успешном завершении установки (рис. 1.5).



Рис. 1.5. Финальное сообщение после установки Python

## 1.1.2. Установка Python в Linux

Чаще всего интерпретатор Python уже входит в состав дистрибутива Linux. Это можно проверить, набрав в окне терминала команду:

> python

или

> python3

В первом случае вы запустите Python 2, во втором — Python 3. В будущем, скорее всего, во все дистрибутивы Linux, включающие Python, будет входить только третья его версия. Если у вас при попытке запустить Python выдается сообщение о том, что он не установлен или установлен, но не тот, который вам необходим, то у вас есть возможность взять его из репозитория.

Для установки Python из репозитория Ubuntu воспользуйтесь командой:

> sudo apt-get install python3

## 1.1.3. Проверка интерпретатора Python

Для начала протестируем интерпретатор в командном режиме. Если вы работаете в Windows, то нажмите комбинацию клавиш <Win> + <R> и в открывшемся окне введите: python. В Linux откройте окно терминала и в нем введите: python3 (или python).

В результате Python запустится в командном режиме. Выглядеть это будет примерно так, как показано на рис. 1.6 (иллюстрация приведена для Windows, в Linux результат будет аналогичным).

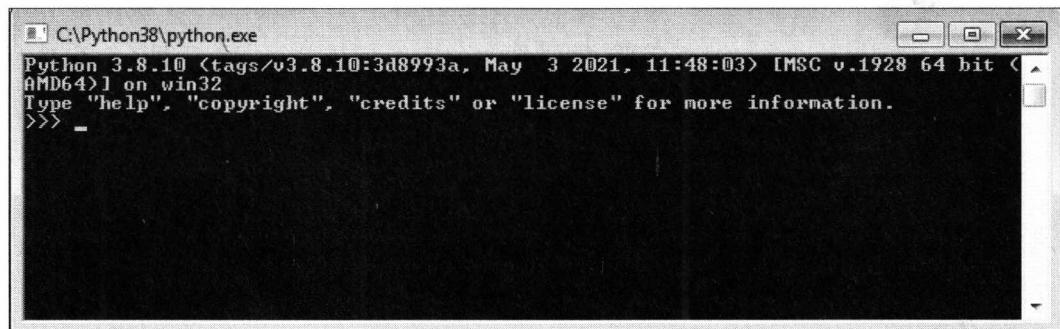


Рис. 1.6. Результат запуска интерпретатора Python в окне терминала

В этом окне введите программный код следующего содержания:

```
print("Hello, World!")
```

В результате вы увидите следующий ответ (рис. 1.7).

Получение такого результата означает, что установка интерпретатора Python прошла без ошибок.

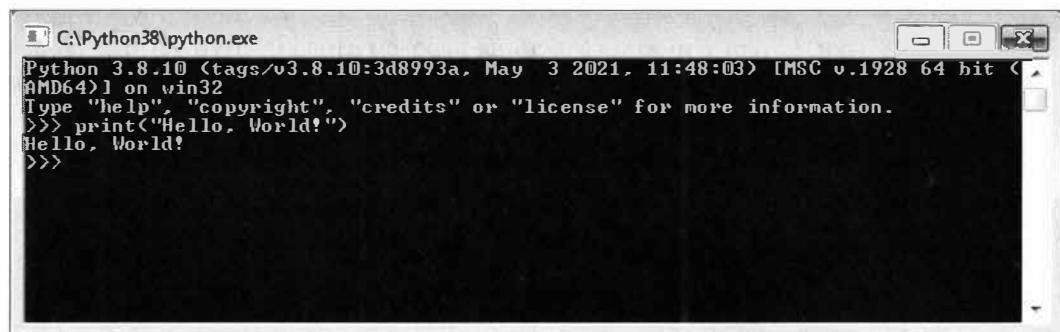


Рис. 1.7. Результат работы программы на Python в окне терминала

## 1.2. Интерактивная среда разработки программного кода PyCharm

В процессе разработки программных модулей удобнее работать в интерактивной среде разработки (IDE), а не в текстовом редакторе. Для Python одним из лучших вариантов считается IDE PyCharm от компании JetBrains. Для скачивания его дистрибутива перейдите по ссылке: <https://www.jetbrains.com/pycharm/download/> (рис. 1.8).

Эта среда разработки доступна для Windows, Linux и macOS. Существуют два вида лицензии PyCharm: **Professional** и **Community**. Мы будем использовать версию



Рис. 1.8. Главное окно сайта для скачивания дистрибутива PyCharm

**Community**, поскольку она бесплатная и ее функционала более чем достаточно для наших задач. На момент подготовки этой книги была доступна версия PyCharm 2022.3.

### 1.2.1. Установка PyCharm в Windows

1. Запустите на выполнение скачанный дистрибутив PyCharm (рис. 1.9).
2. Выберите путь установки программы (рис. 1.10).
3. Укажите ярлыки, которые нужно создать на рабочем столе (запуск 32- или 64-разрядной версии PyCharm), и отметьте флажком опцию .py в области **Create assoca-**

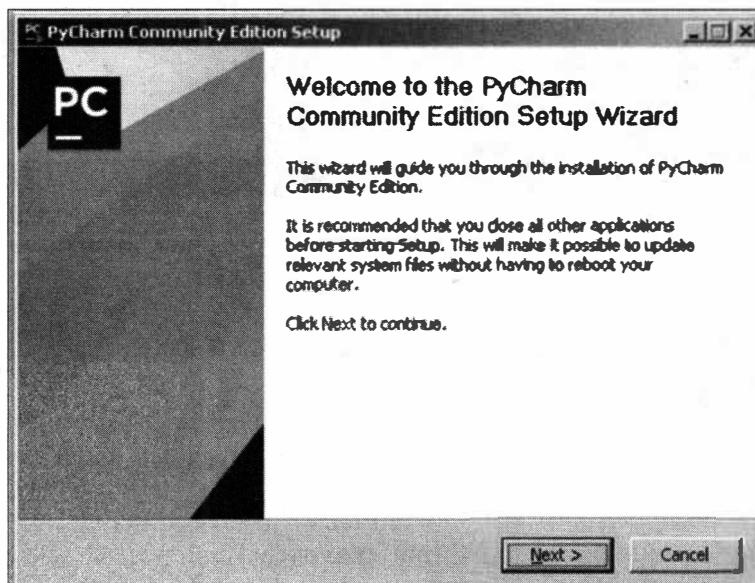


Рис. 1.9. Начальная заставка при инсталляции PyCharm

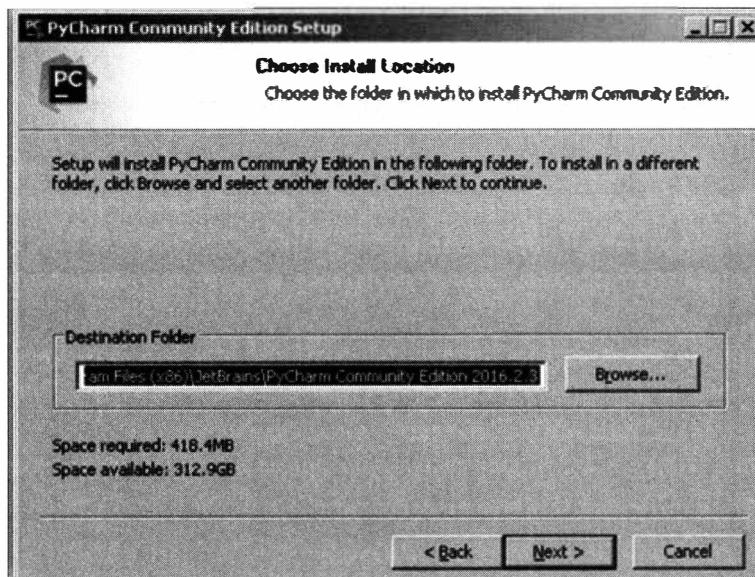


Рис. 1.10. Выбор пути установки PyCharm

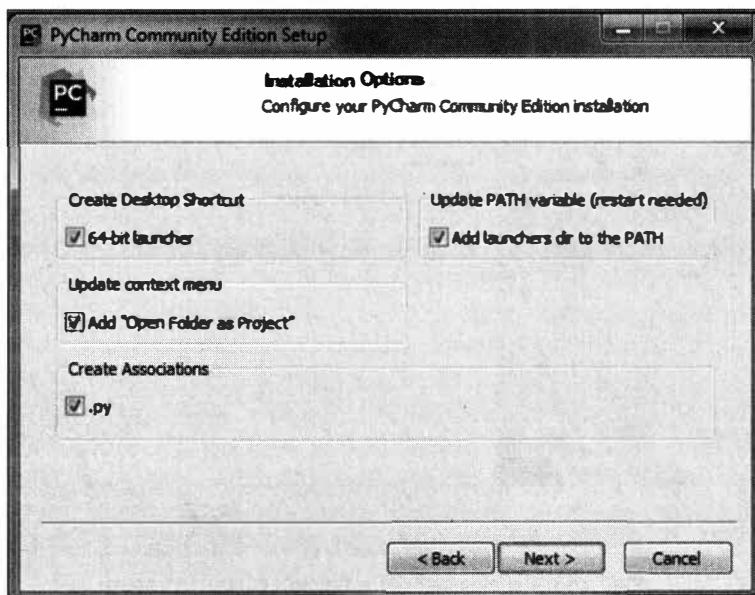


Рис. 1.11. Выбор разрядности устанавливаемой среды разработки PyCharm

**cifications**, если требуется ассоциировать с PyCharm файлы с расширением `py` (рис. 1.11).

4. Выберите имя для папки в меню Пуск (рис. 1.12).
5. Далее PyCharm будет установлен на ваш компьютер (рис. 1.13).



Рис. 1.12. Выбор имени папки для PyCharm в меню Пуск

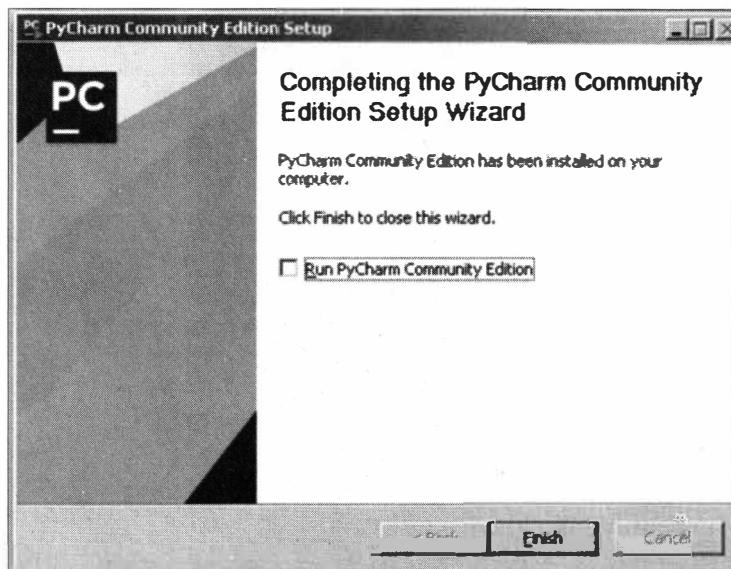


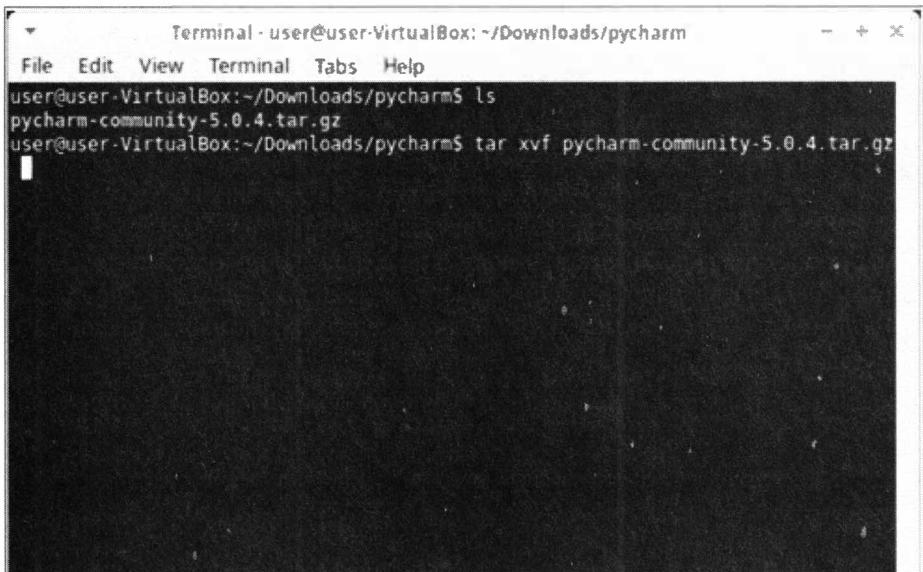
Рис. 1.13. Финальное окно установки пакета PyCharm

## 1.2.2. Установка PyCharm в Linux

1. Скачайте с сайта программы ее дистрибутив на свой компьютер.
2. Распакуйте архивный файл, для чего можно воспользоваться командой:

```
> tar xvf имя_архива.tar.gz
```

Результат работы этой команды представлен на рис. 1.14.



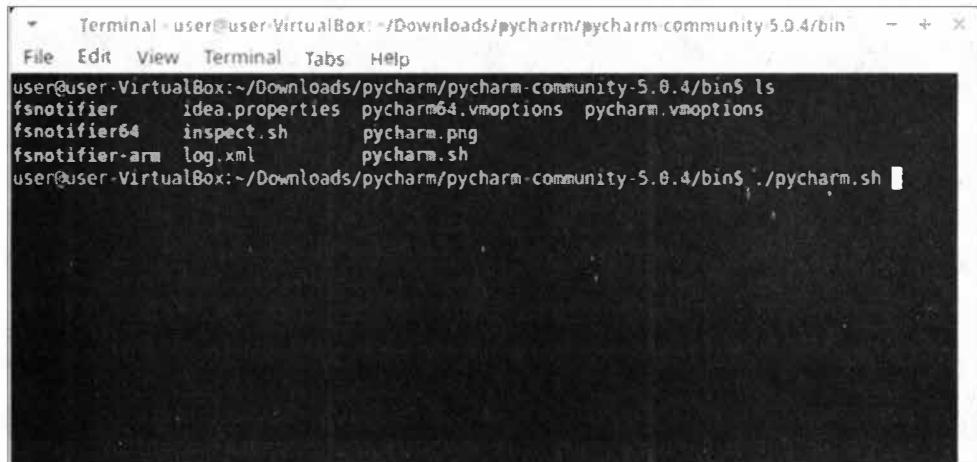
```
Terminal - user@user-VirtualBox: ~/Downloads/pycharm
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Downloads/pycharm$ ls
pycharm-community-5.0.4.tar.gz
user@user-VirtualBox:~/Downloads/pycharm$ tar xvf pycharm-community-5.0.4.tar.gz
```

Рис. 1.14. Результат работы команды распаковки архива PyCharm

3. Перейдите в каталог, который был создан после распаковки дистрибутива, найдите в нем подкаталог bin и зайдите в него. Запустите установку PyCharm командой:

```
> ./pycharm.sh
```

Результат работы этой команды представлен на рис. 1.15.



```
Terminal - user@user-VirtualBox: ~/Downloads/pycharm/pycharm-community-5.0.4/bin
File Edit View Terminal Tabs Help
user@user-VirtualBox:~/Downloads/pycharm/pycharm-community-5.0.4/bin$ ls
fsnotifier  idea.properties  pycharm64.vmoptions  pycharm.vmoptions
fsnotifier64 inspect.sh      pycharm.png
fsnotifier-arm log.xml       pycharm.sh
user@user-VirtualBox:~/Downloads/pycharm/pycharm-community-5.0.4/bin$ ./pycharm.sh
```

Рис. 1.15. Результаты работы команды инсталляции PyCharm

### 1.2.3. Проверка PyCharm

1. Запустите PyCharm и выберите вариант **Create New Project** в открывшемся окне (рис. 1.16).

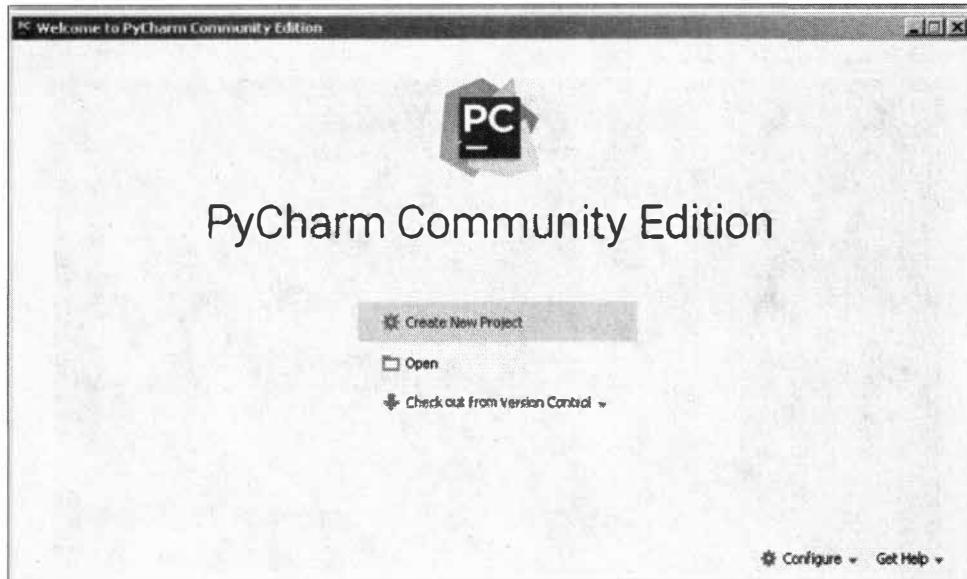


Рис. 1.16. Создание нового проекта в среде разработки PyCharm

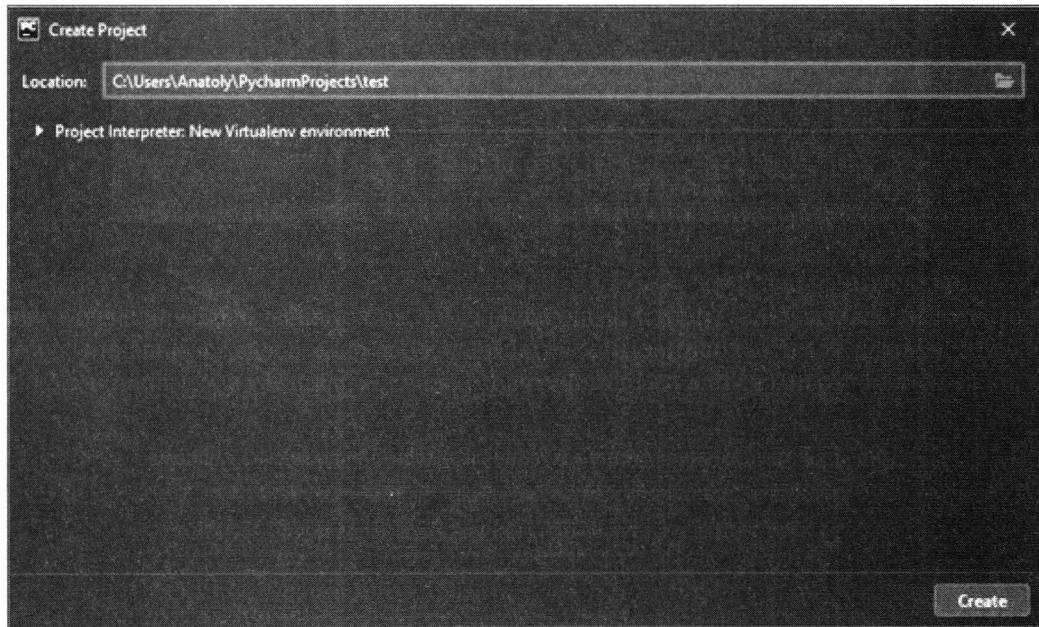


Рис. 1.17. Указание пути до проекта в среде разработки PyCharm

2. Укажите путь до создаваемого проекта Python и интерпретатор, который будет использоваться для его запуска и отладки (рис. 1.17).
3. Добавьте в проект файл, в котором будет храниться программный код Python (рис. 1.18).
4. Введите одну строчку кода программы (рис. 1.19).

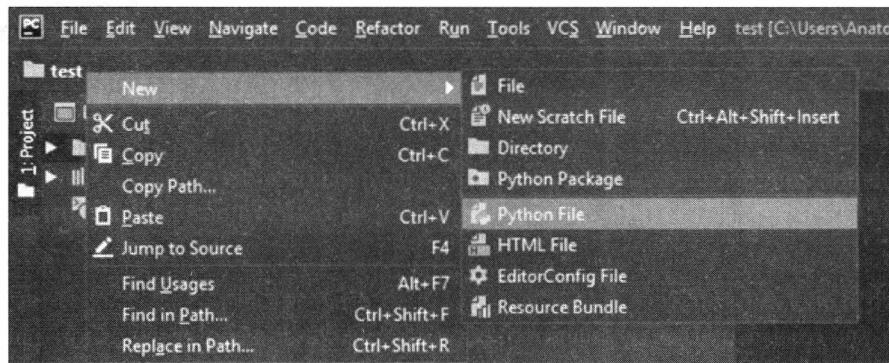


Рис. 1.18. Добавление в проект файла для программного кода на Python

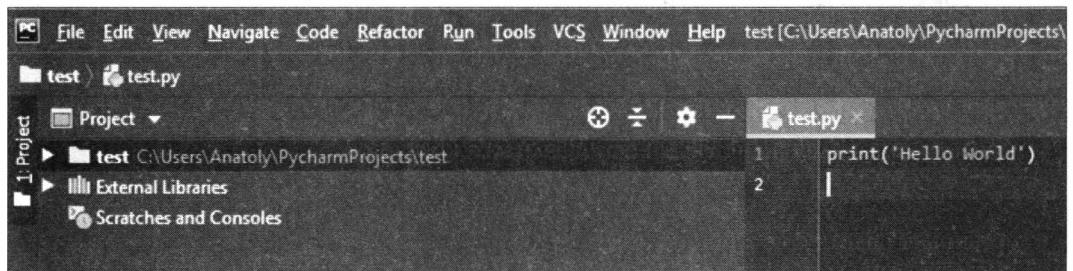


Рис. 1.19. Одна строка программного кода на Python в среде разработки PyCharm

5. Запустите программу командой **Run** (рис. 1.20).

В результате в нижней части экрана должно открыться окно с выводом результатов работы программы (рис. 1.21).

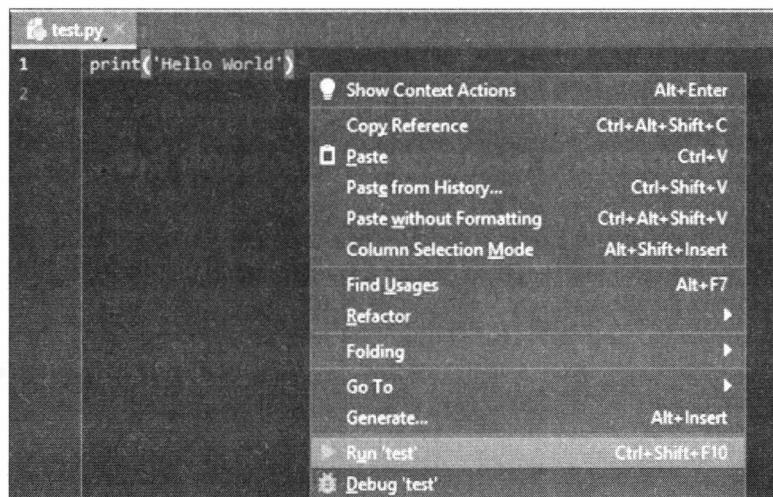


Рис. 1.20. Запуск строки программного кода на Python в среде разработки PyCharm

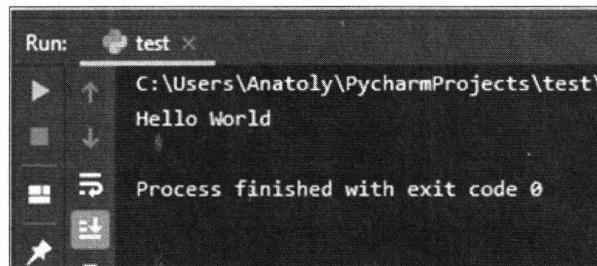


Рис. 1.21. Вывод результатов работы программы на Python в среде разработки PyCharm

## 1.3. Установка пакетов в Python с использованием менеджера пакетов pip

В процессе разработки программного обеспечения на Python часто возникает необходимость воспользоваться пакетом (библиотекой), который в текущий момент отсутствует на вашем компьютере.

В этом разделе вы узнаете о том, откуда можно взять нужный вам дополнительный инструментарий для разработки ваших программ. В частности:

- где взять отсутствующий пакет;
- как установить pip — менеджер пакетов в Python;
- как использовать pip;
- как установить пакет;
- как удалить пакет;
- как обновить пакет;
- как получить список установленных пакетов;
- как выполнить поиск пакета в репозитории.

### 1.3.1. Репозиторий пакетов программных средств PyPI

Необходимость в установке дополнительных пакетов возникает довольно часто, поскольку решение практических задач обычно выходит за рамки базового функционала, который предоставляет Python. Это, например, создание веб-приложений, обработка изображений, распознавание объектов, нейронные сети и другие элементы искусственного интеллекта, геолокация и т. п. В таком случае необходимо узнать, какой пакет содержит функционал, который вам необходим, найти его, скачать, разместить в нужном каталоге и начать использовать. Все указанные действия можно выполнить вручную, однако этот процесс поддается автоматизации. К тому же скачивать пакеты с неизвестных сайтов может быть весьма опасно.

В рамках Python все эти задачи автоматизированы и решены. Существует так называемый Python Package Index (PyPI) — репозиторий, открытый для всех разработчиков на Python, в котором вы можете найти пакеты для решения практически любых задач. При этом у вас отпадает необходимость в разработке и отладке сложного программного ко-

да — вы можете воспользоваться уже готовыми и проверенными решениями огромного сообщества программистов на Python. Вам нужно просто подключить требуемый пакет или библиотеку к своему проекту и активировать уже реализованный в них функционал. В этом и заключается преимущество Python перед другими языками программирования, когда небольшим количеством программного кода можно реализовать решение достаточно сложных практических задач. Там также есть возможность выкладывать свои пакеты. Для скачивания и установки нужных модулей в ваш проект существует специальная утилита, которая называется pip. Сама аббревиатура, которая на русском языке звучит как «пип», фактически раскрывается как «установщик пакетов» или «предпочитаемый установщик программ». Это утилита командной строки, которая позволяет устанавливать, переустанавливать и деинсталлировать пакеты PyPI простой командой pip.

### 1.3.2. pip — менеджер пакетов в Python

pip — утилита консольная (без графического интерфейса). После того как вы ее скачаете и установите, она пропишется в PATH и будет доступна для использования. Этую утилиту можно запускать самостоятельно, например через терминал Windows или в терминальном окне PyCharm командой:

```
> pip <аргументы>
```

pip можно запустить и через интерпретатор Python:

```
> python -m pip <аргументы>
```

Ключ -m означает, что мы хотим запустить модуль (в нашем случае pip).

При развертывании современной версии Python (начиная с Python 2.7.9 и более поздних версий) pip устанавливается автоматически. В PyCharm проверить наличие модуля pip достаточно просто — для этого нужно войти в настройки проекта через меню **File | Settings | Project Interpreter**. Модуль pip должен присутствовать в списке загруженных пакетов и библиотек (рис. 1.22).

При отсутствии в списке этого модуля последнюю его версию можно загрузить, нажав на значок + в правой части окна и выбрав модуль pip из списка (рис. 1.23).

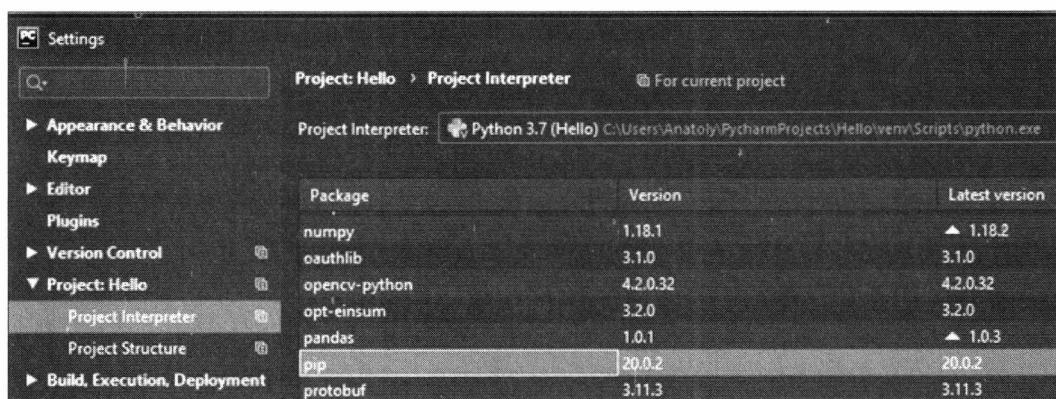


Рис. 1.22. Проверка наличия в проекте модуля pip

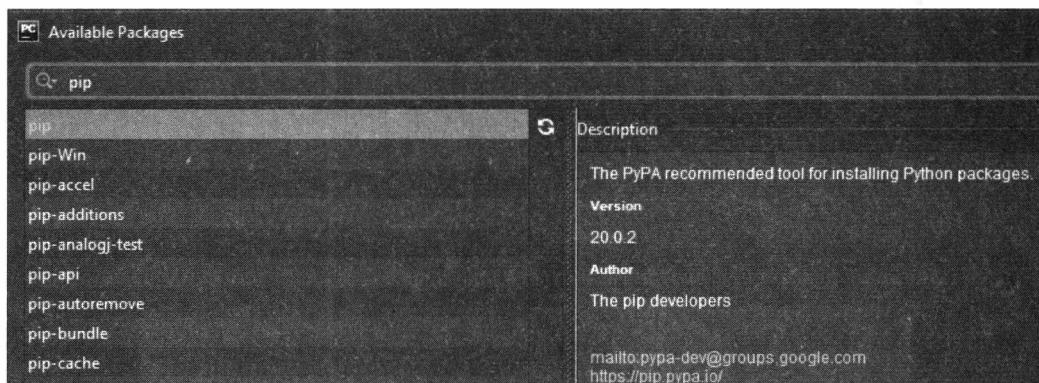


Рис. 1.23. Загрузка модуля pip

### 1.3.3. Использование менеджера пакетов pip

Здесь мы рассмотрим основные варианты использования pip: установку, удаление и обновление пакетов.

Pip позволяет установить самую последнюю версию пакета, конкретную версию или воспользоваться логическим выражением, через которое можно определить, что вам, например, нужна версия не ниже указанной. Также есть поддержка установки пакетов из репозитория. Рассмотрим эти варианты (здесь Name — это имя пакета).

- Установка последней версии пакета:

```
> pip install Name
```

- Установка определенной версии:

```
> pip install Name==3.2
```

- Установка пакета с версией не ниже 3.1:

```
> pip install Name>=3.1
```

- Для того чтобы удалить пакет, воспользуйтесь командой:

```
> pip uninstall Name
```

- Для обновления пакета задайте ключ --upgrade:

```
> pip install --upgrade Name
```

- Для вывода списка всех установленных пакетов служит команда:

```
> pip list
```

- Если вы хотите получить более подробную информацию о конкретном пакете, то укажите аргумент show:

```
> pip show Name
```

- Если вы не знаете точного названия пакета или хотите посмотреть на пакеты, содержащие конкретное слово, то вы можете это сделать, используя аргумент search:

```
> pip search "test"
```

Если вы запускаете pip в терминале Windows, то терминальное окно автоматически закроется после того, как эта утилита завершит свою работу. При этом вы просто не успеете увидеть результаты ее работы. Чтобы терминальное окно не закрывалось автоматически, команды pip нужно запускать в нем с ключом /k. Например, запуск процедуры установки пакета tensorflow должен выглядеть так, как показано на рис. 1.24.

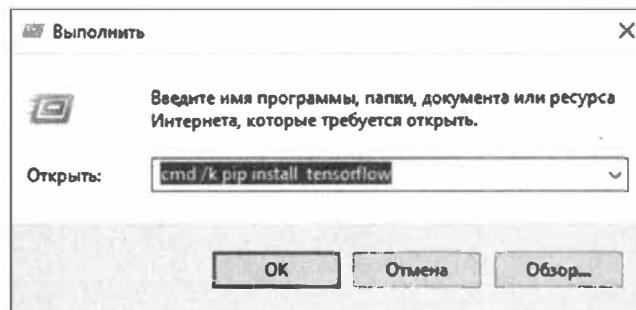


Рис. 1.24. Выполнение команды модуля pip в терминальном окне Windows

Если же пакет pip запускается из терминального окна PyCharm, то в указании дополнительных ключей нет необходимости, т. к. терминальное окно после завершения работы программ не закрывается. Пример выполнения той же команды в терминальном окне PyCharm показан на рис. 1.25.

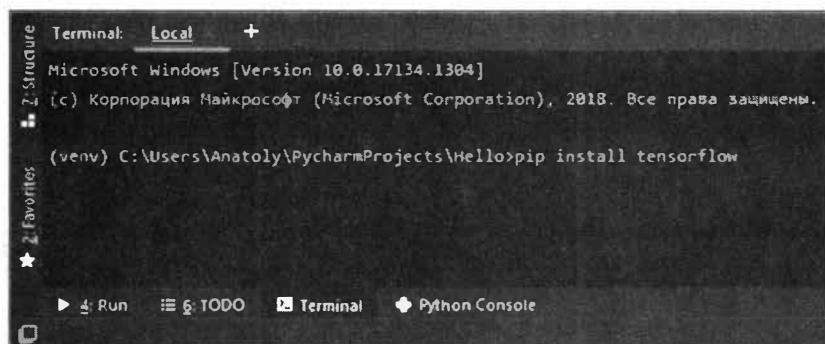


Рис. 1.25. Выполнение команды pip в терминальном окне PyCharm

## 1.4. Фреймворк Django для разработки веб-приложений

Итак, основной инструментарий для разработки программ на языке Python установлен, и мы можем перейти к установке дополнительных библиотек. В этом разделе мы установим фреймворк Django, который позволяет разрабатывать веб-приложения.

Запустим среду разработки PyCharm и создадим в ней новый проект web\_1. Для этого в главном меню выберите опцию File | New Project (рис. 1.26).

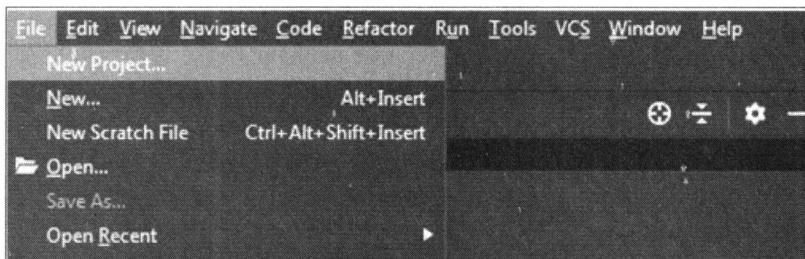


Рис. 1.26. Создание нового проекта в среде разработки PyCharm

Откроется окно, где вы можете задать имя создаваемому проекту, определить виртуальное окружение для этого проекта и указать каталог, в котором находится интерпретатор Python. Задайте новому проекту имя Web\_1 и нажмите кнопку **Create** (рис. 1.27).

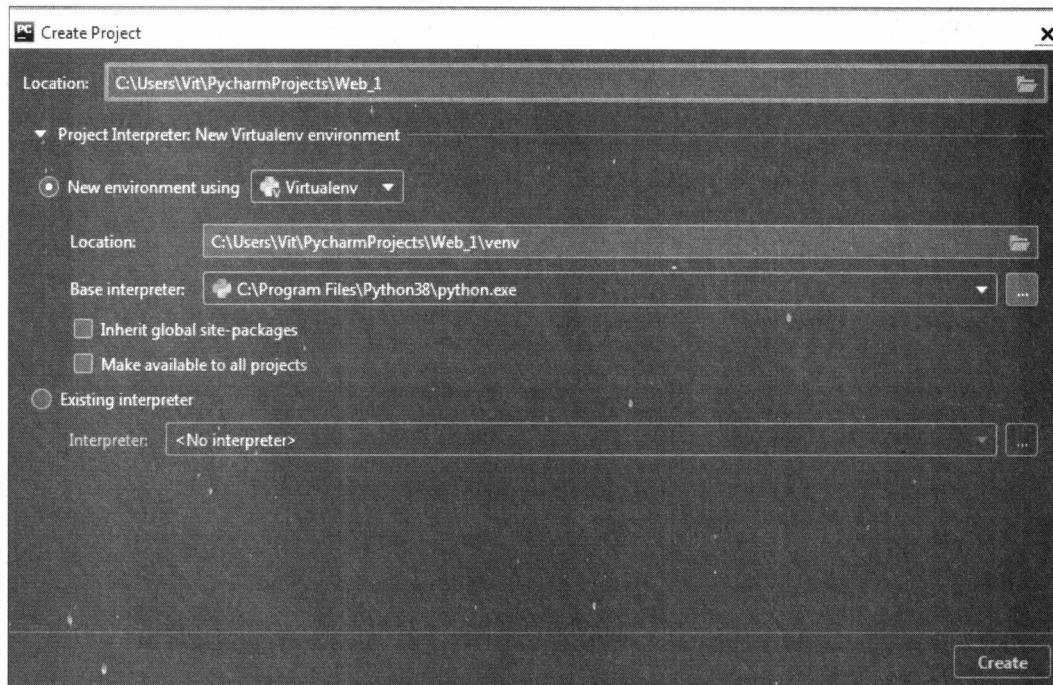


Рис. 1.27. Задаем новому проекту имя Web\_1 в среде разработки PyCharm

Будет создан новый проект. Это, по сути дела, шаблон проекта, в котором пока еще ничего нет (рис. 1.28).

Фреймворк Django — это фактически дополнительная библиотека к Python, и установить данный инструментарий можно так же, как и любую другую библиотеку. Для установки библиотеки Django можно в меню **File** выбрать опцию **Settings...** (рис. 1.29).

В левой части открывшегося окна настроек выберите опцию **Project Interpreter**, и в правой части окна будет показана информация об интерпретаторе языка Python и подключенных к нему библиотеках (рис. 1.30).

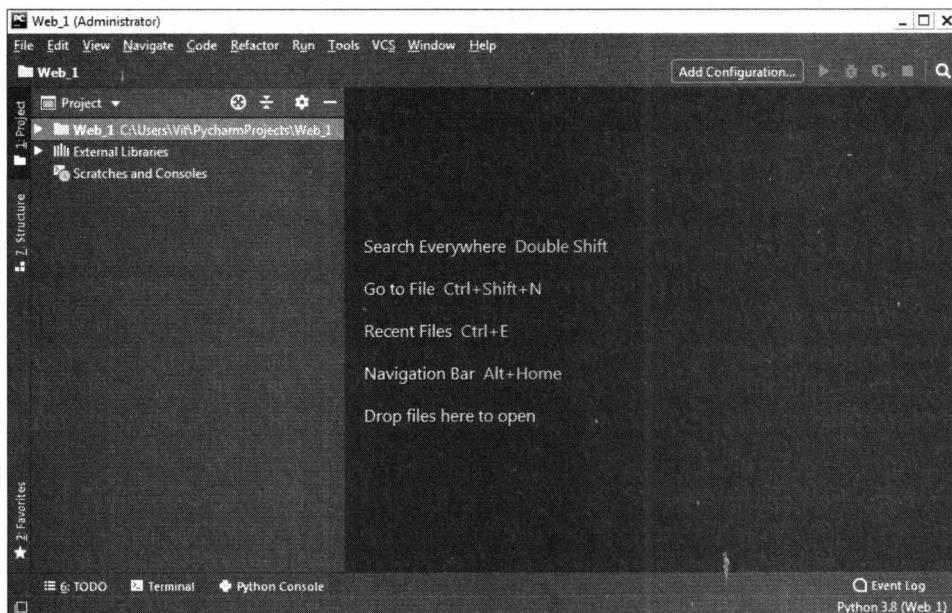


Рис. 1.28. Внешний вид интерфейса PyCharm с пустым проектом

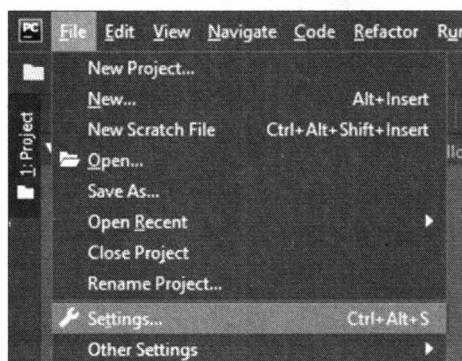


Рис. 1.29. Вызов окна Settings настройки параметров проекта

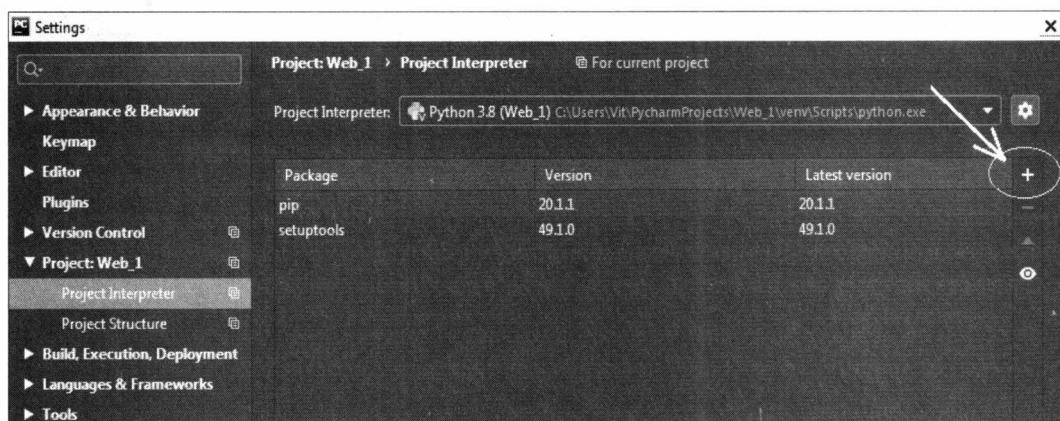


Рис. 1.30. Информация об интерпретаторе языка Python

Чтобы добавить новую библиотеку, нужно нажать на значок в правой части окна, после чего будет отображен полный список доступных библиотек. Здесь можно либо пролистать весь список и найти библиотеку Django, либо набрать наименование этой библиотеки в верхней строке поиска, и она будет найдена в списке (рис. 1.31).

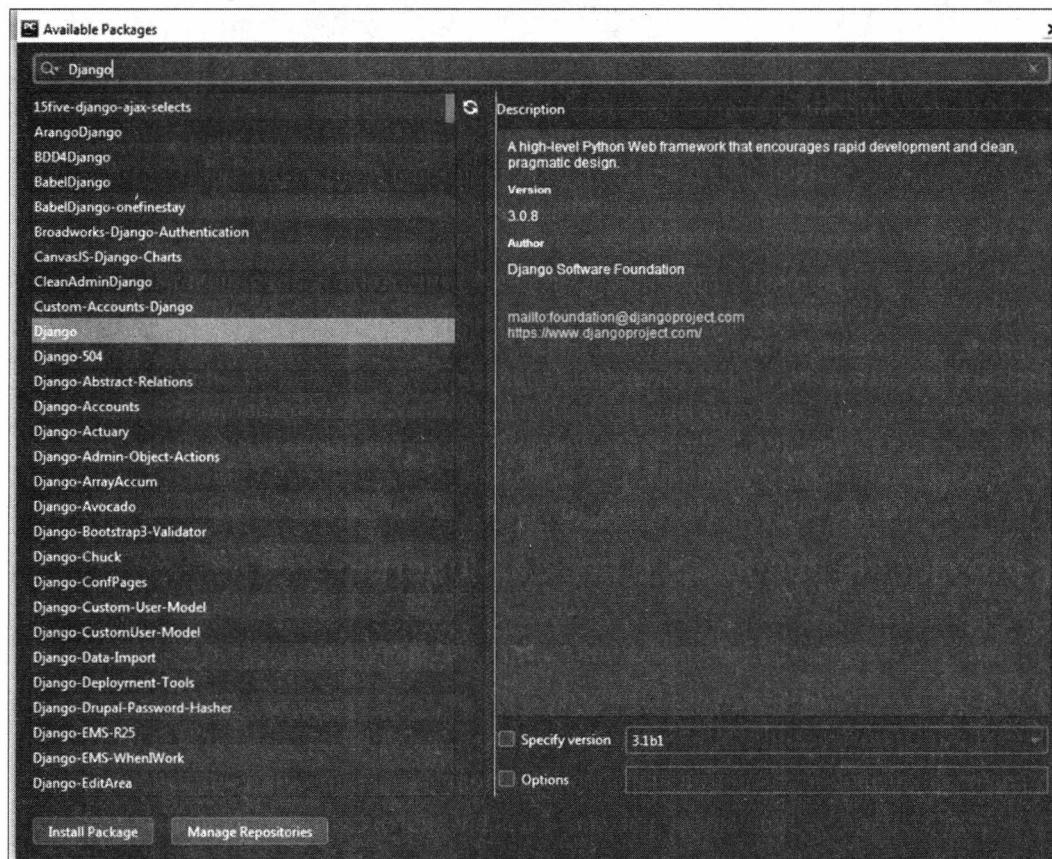


Рис. 1.31. Поиск библиотеки Django в списке доступных библиотек

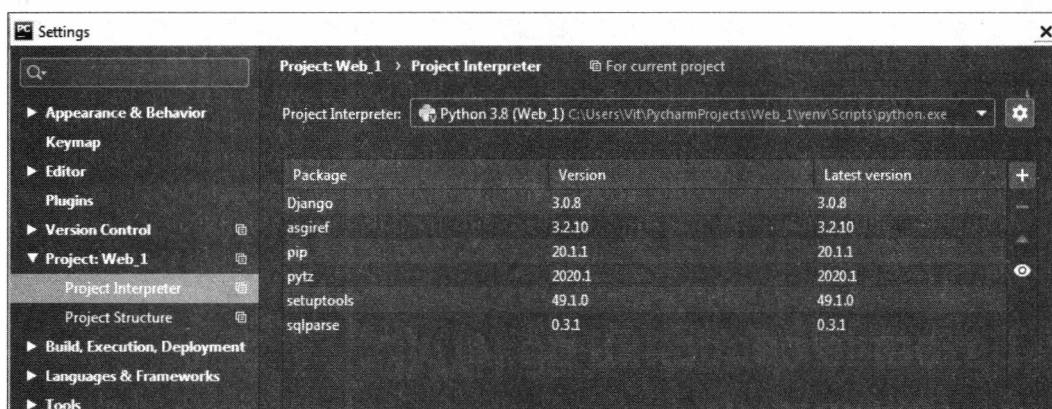


Рис. 1.32. Библиотека Django добавлена в список подключенных библиотек

Нажмите теперь на кнопку **Install Package**, и выбранная библиотека будет добавлена в ваш проект (рис. 1.32).

На момент подготовки данной книги к изданию был доступен фреймворк Django версии 4.1.4. Теперь у нас есть минимальный набор инструментальных средств, который необходим для разработки веб-приложений на языке Python. Впрочем, в процессе рассмотрения конкретных примеров нам понадобится загрузка еще ряда дополнительных пакетов и библиотек. Их описание и процедуры подключения будут представлены в последующих главах.

## 1.5. Менеджер баз данных SQLiteStudio

Поскольку мы планируем создавать сайты, которые взаимодействуют с базами данных (БД), то для этого нам понадобится специальное программное средство. По умолчанию Django использует СУБД SQLite. Скачать установщик менеджера баз данных SQLite для ПК с ОС Windows можно по ссылке <https://sqlitestudio.pl/>.

После запуска загрузочного файла мы увидим, что это программное средство имеет русскоязычный интерфейс (рис. 1.33).

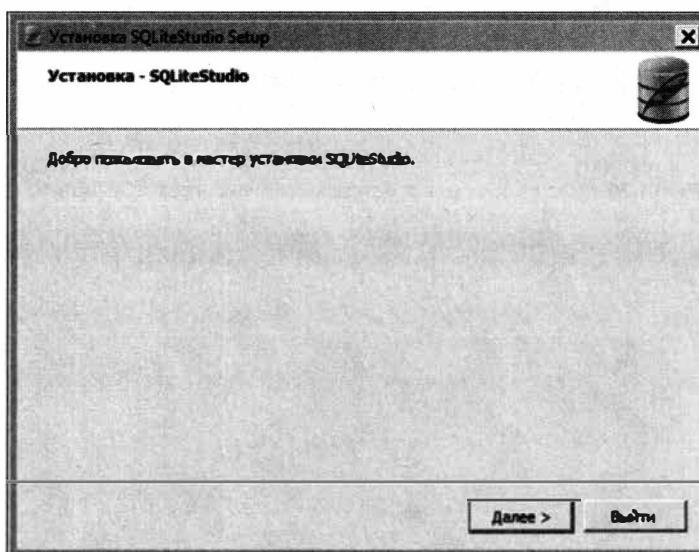


Рис. 1.33. Начальное окно установки менеджера SQLiteStudio

В следующих окнах, которые мы пропустим, следует выполнять рекомендации мастера установки. В завершающем окне нам будет предложено сразу запустить менеджер SQLiteStudio (рис. 1.34).

Нажмите здесь на кнопку **Завершить**, и откроется окно выбора языка интерфейса (рис. 1.35).

Выбираем русский язык и нажимаем на кнопку **OK**. Если процесс установки прошел корректно, то на экран будет выведено главное окно менеджера SQLiteStudio (рис. 1.36).

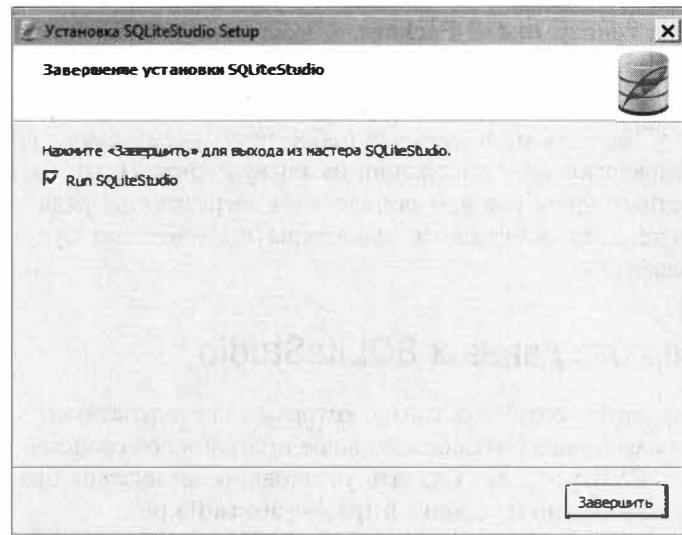


Рис. 1.34. Завершающее окно установки менеджера SQLiteStudio

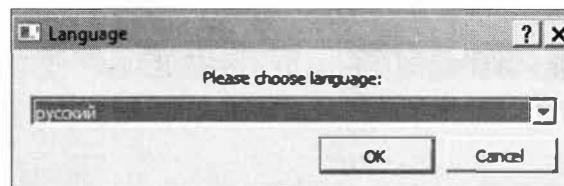


Рис. 1.35. Окно выбора языка интерфейса менеджера SQLiteStudio

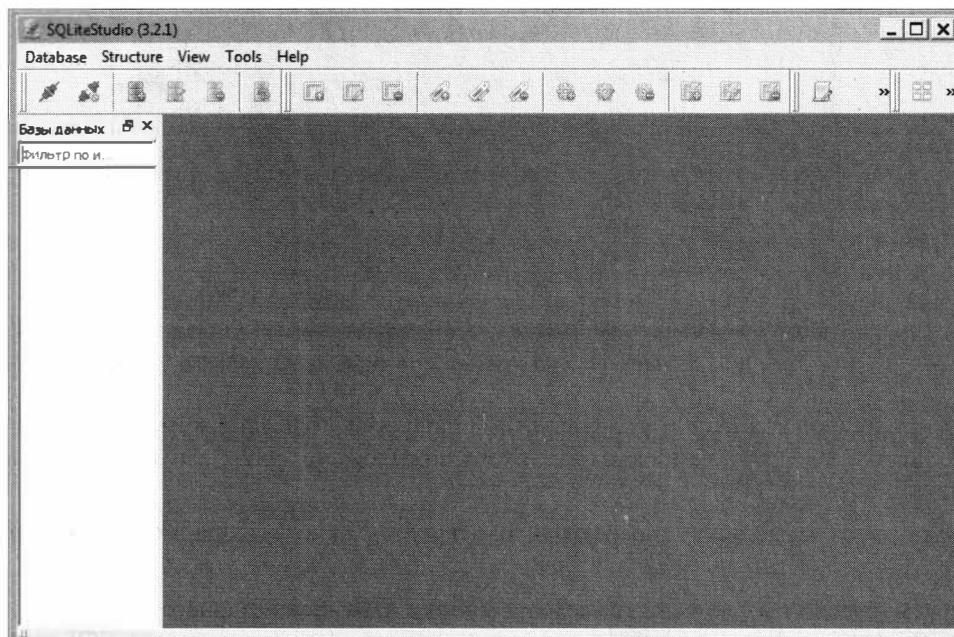


Рис. 1.36. Главное окно менеджера SQLiteStudio

Как можно видеть, несмотря на выбор русского языка в окне выбора языка интерфейса (см. рис. 1.35), меню в главном окне менеджера SQLiteStudio выведено в англоязычном исполнении. Это, скорее всего, не совсем корректная локализация именно данной версии программного продукта — часть интерфейса осталась нелокализованной.

## 1.6. Краткие итоги

В этой главе мы познакомились с основными инструментальными средствами, с помощью которых можно разрабатывать веб-приложения. Это интерпретатор Python, интерактивная среда разработки программного кода PyCharm, фреймворк Django для разработки веб-приложений и менеджер работы с базами данных SQLiteStudio. Установив на свой компьютер перечисленные инструментальные средства, теоретически можно приступать к написанию программного кода и созданию HTML-страниц. Мы пока не устанавливали фреймворк Bootstrap и сервер баз данных MySQL, поскольку этому инструментарию будут посвящены отдельные главы. Наличие инструмента — это необходимое, но недостаточное условие для того, чтобы приступить к программированию. Нужно еще и уметь применять этот инструментарий.

Те специалисты, которые имеют достаточный опыт программирования на Python, могут сразу перейти к следующей главе, в которой дается некоторое представление о веб-технологиях и языке разметки HTML. Для тех же, кто только начинает знакомиться с миром Python, нужно получить хотя бы элементарные навыки работы с этим языком программирования, обратившись к предназначенной для этих целей специальной литературе.

Итак, переходим к следующей главе и знакомимся с технологиями веб-программирования.



## ГЛАВА 2



# Веб-технологии и базовые сведения об HTML

В настоящее время веб-технологии стремительно развиваются, проникая в самые разнообразные сферы нашей жизни. Для компаний присутствие в сети Интернет — это возможность рассказать о своих товарах и услугах, найти потенциальных партнеров и клиентов, снизить издержки за счет интернет-торговли и «облачных» сервисов. Рядовые пользователи уже привыкли к услугам интернет-магазинов, интернет-банкинга, общаются в социальных сетях, могут получать через Интернет государственные услуги.

Из материалов этой главы вы также узнаете:

- что такое веб-технологии;
- в чем различия технологий клиентского и серверного программирования;
- какие фреймворки предназначены для создания веб-приложений на Python;
- какие теги помогут представить текст на HTML-страницах;
- что такое каскадные таблицы стилей;
- какие возможности таит в себе скриптовый язык JavaScript.

## 2.1. Базовые сведения о веб-технологиях

Под веб-технологиями в дальнейшем мы будем понимать всю совокупность средств для организации взаимодействия пользователей с удаленными приложениями. Поскольку в каждом сеансе взаимодействуют две стороны (сервер и клиент), то и веб-приложения можно разделить на две группы: приложения на стороне сервера (*server-side*) и приложения на стороне клиента (*client-side*). Благодаря веб-приложениям удаленному пользователю доступны не только статические документы, но и сведения из базы данных. Использование баз данных в Интернете приобрело огромную популярность и практически стало отдельной отраслью компьютерной технологии.

Но для начала разберемся с основными понятиями веб-технологий: что такое веб-сайт и веб-страница. Часто неопытные пользователи неправомерно смешивают эти понятия. *Веб-страница* — это минимальная логическая единица в сети Интернет, которая представляет собой документ, однозначно идентифицируемый уникальным интернет-адресом (URL, Uniform Resource Locator — унифицированный указатель ресурса). *Веб-*

*сайт* — это набор тематически связанных веб-страниц, находящихся на одном сервере и принадлежащих одному владельцу. В частном случае веб-сайт может состоять из единственной веб-страницы. Сеть Интернет является совокупностью всех веб-сайтов.

Для формирования веб-страниц используется язык разметки гипертекста HTML (Hyper Text Markup Language). С его помощью осуществляется логическая (смысловая) разметка документа (веб-страницы). Для управления внешним видом веб-страниц применяются каскадные таблицы стилей (от англ. Cascading Style Sheets, CSS). Сформированные на сервере HTML-документы можно просмотреть на компьютере клиента с помощью специальной программы — *браузера*.

Как было отмечено ранее, совокупность связанных между собой веб-страниц представляет собой веб-сайт. Сайты можно условно разделить на статические и динамические.

**Статический сайт** — это сайт с неизменным информационным наполнением. Основу статического сайта составляют веб-страницы с постоянным содержимым, разработанные на основе стандартной HTML-технологии. Страницы сайта хранятся в виде HTML-кода в файловой системе сервера. Естественно, на таком сайте могут присутствовать различные видеоролики и анимация. Основная отличительная особенность статического сайта заключается в том, что веб-страницы такого сайта создаются заранее. Для редактирования содержимого страниц и обновления сайта страницы модифицируют вручную с применением HTML-редактора и затем заново загружают на сайт. Схема взаимодействия клиента со статическим сайтом приведена на рис. 2.1.

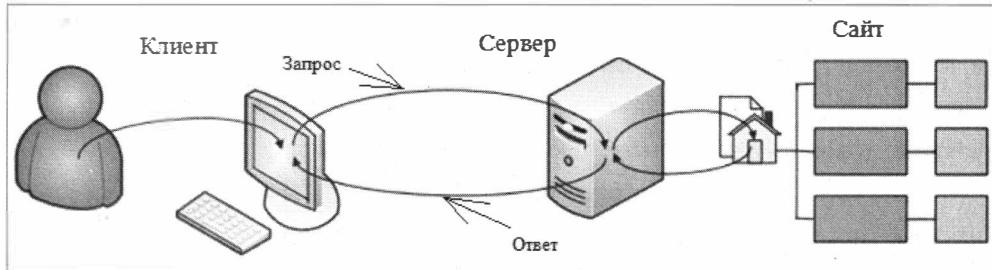


Рис. 2.1. Схема взаимодействия клиента со статическим сайтом

Такая схема приемлема в том случае, когда содержимое сайта довольно постоянно и изменяется сравнительно редко. Если же информация, размещенная на статическом сайте, требует регулярной актуализации и обновления, то для его поддержания неизбежны внушительные трудозатраты. Таким образом, статический сайт дешевле в разработке и технической поддержке, но эти достоинства могут нивелироваться серьезными недостатками, связанными с необходимостью оперативного обновления актуальной информации и достаточно высокой трудоемкостью модификации.

**Динамический сайт** — это сайт с динамическим информационным наполнением. Динамические страницы также формируются с помощью HTML, но такие страницы обновляются постоянно, нередко при каждом новом обращении к ним. Динамические сайты основываются на статических данных и HTML-разметке, но дополнительно содержат программную часть (скрипты), а также базу данных (БД), благодаря которым страница «собирается» из отдельных фрагментов в режиме реального времени. Это по-

зволяет обеспечить гибкость в подборе и представлении информации, соответствующей конкретным запросам посетителей сайта.

Таким образом, динамический сайт состоит из набора различных блоков: шаблонов страниц, информационного наполнения (контента) и скриптов, хранящихся в виде отдельных файлов. Иными словами, динамическая веб-страница формируется из страницы-шаблона и добавляемых в шаблон динамических данных. Тип данных, загружаемых в шаблон, будет зависеть от того, какой запрос сделал клиент. Схема взаимодействия клиента с динамическим сайтом приведена на рис. 2.2.

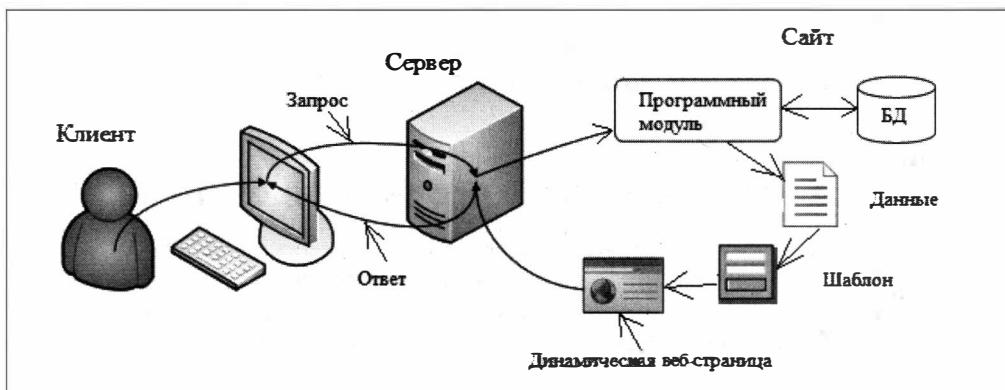


Рис. 2.2. Схема взаимодействия клиента с динамическим сайтом

Динамичность сайта заключается в том, что для изменения страницы достаточно поменять ее информационное наполнение, а сам механизм формирования и вывода страницы остается тем же. Кроме получения различных данных, удаленный клиент может изменить и содержание информационной части сайта. Для этого используются *веб-формы*. Клиент заполняет веб-форму своими данными, и эта информация вносится в БД сайта.

Динамические сайты различаются в зависимости от применяемых технологий и процесса получения динамических страниц. Динамические страницы можно получить следующими способами:

- генерацией страницы на стороне сервера, осуществляющей серверными скриптами на языках PHP, Perl, ASP.NET, Java, Python и др. При этом информационное наполнение страниц хранится в базах данных;
- генерацией страницы на стороне клиента (JavaScript);
- комбинированной генерацией. Чаще всего на практике встречается именно комбинация первых двух способов.

## 2.1.1. Технологии клиентского программирования

Простейшим средством «оживления» веб-страниц, добавления динамических эффектов и задания реакции на пользовательские действия является скриптовый язык программирования JavaScript. Сценарии (скрипты) JavaScript внедряются непосредственно в веб-страницу или связываются с ней и после загрузки страницы с сервера выполня-

ются браузером на стороне клиента. Все современные браузеры имеют поддержку JavaScript.

JavaScript — это компактный объектно-ориентированный язык для создания клиентских веб-приложений. Этот язык применяется для обработки событий, связанных с вводом и просмотром информации на веб-страницах. JavaScript обычно используется в клиентской части веб-приложений, в которых клиентом выступает браузер, а сервером — удаленный веб-сервер. Обмен информацией в веб-приложениях происходит по сети. Одним из преимуществ такого подхода является тот факт, что клиенты не зависят от конкретной операционной системы пользователя, поэтому веб-приложения технологии клиентского программирования представляют собой кросс-платформенные сервисы. Язык сценариев JavaScript позволяет создавать интерактивные веб-страницы и содержит средства управления окнами браузера, элементами HTML-документов и стилями CSS.

## 2.1.2. Технологии серверного программирования

Без серверного программирования нельзя обойтись, если необходимо изменять и сохранять какую-либо информацию, хранящуюся на сервере (например, организовать прием и сохранение сообщений, отправляемых пользователями). Без серверных скриптов невозможно представить себе гостевые книги, форумы, чаты, опросы (голосования), счетчики посещений и другие программные компоненты, которые активно взаимодействуют с базами данных. Серверное программирование позволяет решать такие задачи, как регистрация и авторизация пользователей, управление аккаунтом (в почтовых веб-системах, социальных сетях и др.), поиск информации в базе данных, работа интернет-магазина и т. п. Серверные языки программирования открывают перед программистом большие функциональные возможности. Современные сайты зачастую представляют собой чрезвычайно сложные программно-информационные системы, решающие разнообразные задачи, и теоретически могут дублировать функции большинства бизнес-приложений.

Работа серверных скриптов зависит от платформы, т. е. от того, какие технологии поддерживаются сервером, на котором расположен сайт. Например, основной технологией, поддерживаемой компанией Microsoft, является ASP.NET (Active Server Pages, активные серверные страницы). Другие серверы могут поддерживать языки Perl, PHP, Python.

- **Perl** — высокоуровневый интерпретируемый динамический язык программирования общего назначения. Он является одним из наиболее старых языков, используемых для написания серверных скриптов. По популярности Perl сейчас уступает более простому в освоении языку PHP. В настоящее время используются бесплатные технологии EmbPerl и mod\_perl, позволяющие обрабатывать HTML-страницы со вставленными скриптами на языке Perl.
- **PHP (Personal Home Page)** — язык сценариев общего назначения, в настоящее время интенсивно применяемый для разработки веб-приложений. PHP-код может внедряться непосредственно в HTML. Это язык с открытым кодом, крайне простой для освоения, но вместе с тем способный удовлетворить запросы профессиональных веб-программистов, т. к. имеет большой набор специализированных встроенных средств.

- **Python** — универсальный язык программирования, применимый, в том числе и для разработки веб-приложений. Важно, что в Python приложение постоянно находится в памяти, обрабатывая множество запросов пользователей без «перезагрузки». Таким образом, поддерживается правильное предсказуемое состояние приложения. В зависимости от того, какой фреймворк будет использоваться совместно с Python, взаимодействия могут существенно упрощаться. Например, Django, который сам написан на Python, имеет систему шаблонов для написания специальных HTML-файлов, которые могут включать код Python и взаимодействовать с данными из СУБД.

### 2.1.3. Фреймворки Django и Bootstrap для разработки веб-приложений

Любое веб-приложение состоит из клиентской части и серверной части, которые тесно связаны между собой (рис. 2.3).

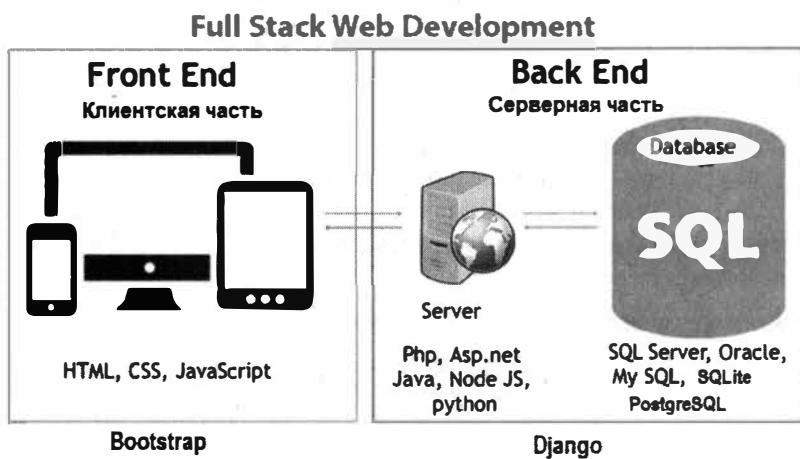


Рис. 2.3. Структура веб-приложения с фреймворками Django и Bootstrap

Каждая из этих частей разрабатывается с использованием разного набора инструментов, соответственно и к программистам, разрабатывающим ту или иную часть приложения, предъявляются различные требования. Учитывая эти особенности, различают три типа специалистов, занимающихся разработкой веб-приложений:

- разработчики клиентской части приложения (frontend-разработчики);
  - разработчики серверной части приложения (backend-разработчики);
  - разработчики полного стека (full stack web development).
- Остановимся на различиях этих направлений, их особенностях и применяемых инструментальных средствах.

**Понятие frontend-разработки.** Под понятием frontend подразумевается разработка видимого для пользователя интерфейса и всех функций, с которыми он может взаимодействовать. По сути, когда вы переходите на любой сайт, то видите там кнопки, текст, различные изображения, анимацию и другие составляющие — все это реализовано при

помочи инструментария frontend. Эти элементы создаются на трех разных языках: HTML, CSS и JavaScript.

Основной инструмент в этой сфере — язык гипертекстовой разметки HTML. Он нужен в основном для разметки документа, т. е. HTML-страниц. С помощью него разработчик создает структуру, добавляет заголовки, списки и форматирует контент.

В связке с языком HTML используется и язык CSS (Cascading Style Sheets), который отвечает за внешний вид страницы. С его помощью оформляются цвета, шрифты и фоны различных блоков. Если выразиться простыми словами, то CSS служит для красивого оформления страницы и настройки ее внешнего вида уже после того, как основная структура была написана при помощи HTML.

С помощью JavaScript (JS) реализуется выполнение различных действий на странице, т. е. добавляется анимация и отклик на запросы пользователя. Например, страница реагирует на перемещение курсора и нажатие на кнопки мыши, изменяя поведение элементов в соответствии с действиями пользователя. Благодаря JS осуществляются действия на стороне клиента без необходимости перезагрузки страницы.

**Backend-разработка.** Под понятием Backend подразумевается разработка бизнес-логики веб-приложения. Эта часть отвечает за взаимодействие пользователя с данными, которые потом отображает клиентская часть приложения. Попросту говоря, это то, что скрыто от глаз пользователя и происходит на удаленном сервере вне его браузера и компьютера. Как только поступает запрос от пользователя (например, когда к элементу интерфейса подведен курсор и нажата кнопка мыши), сигнал сразу же отправляется на сервер, где и обрабатывается для дальнейшего вывода информации на экран. Это и есть логика сайта, заключающаяся в трех простых шагах:

1. Получение запроса от пользователя на сервер.
2. Обработка запроса пользователя на сервере.
3. Отправка ответа пользователю в его браузер.

При создании клиентской части приложения разработчики всегда используют описанные ранее языки программирования: HTML, CSS и JavaScript. Как правило, эти языки объединяются в рамках одного инструментального средства, среди которых, пожалуй, наиболее популярен фреймворк Bootstrap, тесно интегрированный с Django. С его помощью мы и будем разрабатывать клиентскую часть веб-приложений.

С разработкой серверной части дело обстоит немного иначе. Выбор языка и инструментария зависит от сервера. Но, как правило, разработчики выбирают один из универсальных языков программирования: Java, PHP, Python, Ruby и др.

В данной книге мы будем использовать язык программирования Python и следующие фреймворки:

- Django для разработки серверной части веб-приложений;
- Bootstrap для разработки клиентской части веб-приложений.

Что касается Django, то это инструмент, позволяющий упростить реализацию взаимодействия пользователей с базами данных, логику обработки данных на сервере и формирование динамических веб-страниц на основе шаблонов. Кроме того, административная панель Django дает возможность владельцам сайтов осуществлять удаленное

администрирование развернутых сайтов, их модификацию, управление доступом пользователей и техническую поддержку. Этот фреймворк предназначен для бэкенд-разработчиков, соответственно с его помощью повышается эффективность разработки серверной части приложения, но в паре с ним необходимы другие инструменты для создания привлекательных HTML-страниц.

На сегодняшний день наиболее популярный инструмент для фронтенд-разработчиков — фреймворк Bootstrap, поскольку он позволяет создавать адаптивные веб-приложения. Адаптивность заключается в том, что интерфейс приложения автоматически адаптируется под разрешение экрана того устройства, на котором оно было запущено, а значит, приложение выглядит одинаково хорошо на экране смартфона, планшета и настольного компьютера. Фреймворк Bootstrap использует языки программирования HTML, CSS и JavaScript, на основе которых можно создавать как структуру HTML-страниц, так и красочное их оформление. Это свободно распространяемый инструментарий, который хорошо интегрируется с Django.

Прежде чем переходить к изучению Django и Bootstrap, рассмотрим базовые сведения об HTML, CSS, JavaScript.

## 2.2. Базовые сведения о HTML

Как было отмечено ранее, язык разметки гипертекста HTML является основой для формирования веб-страниц. С помощью HTML осуществляется логическое форматирование документа, и он предназначен только для этих целей.

HTML-документы строятся на основе тегов, которые структурируют документ. Обычно теги бывают парными, т. е. состоят из открывающего и закрывающего тега, хотя бывают и исключения. Имена открывающих тегов заключаются в угловые скобки `< ... >`, а закрывающие теги обрамляются угловыми скобками и знаком слеш `</ ... >`.

Весь HTML-документ обрамляется парными тегами `<html>...</html>`. Кроме того, для обеспечения корректного отображения документа современный стандарт требует наличия одиночного тега `<!DOCTYPE>`, который может иметь следующую структуру:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

В самом простом случае этот одиночный тег выглядит так:

```
<!DOCTYPE html>
```

Сами HTML-документы состоят из заголовка и тела. Заголовок документа описывается парой тегов `<head>...</head>`, а тело документа обрамляется парными тегами `<body>...</body>`. Таким образом, каркас HTML-документа будет иметь следующую структуру (листинг 2.1).

### Листинг 2.1. Структура HTML-документа

```
<!DOCTYPE HTML>
<html>
  <head>
    СОДЕРЖАНИЕ ЗАГОЛОВКА ДОКУМЕНТА
  </head>
```

```
<body>
    СОДЕРЖАНИЕ ТЕЛА ДОКУМЕНТА
</body>
</html>
```

Заголовок может включать в себя несколько специализированных тегов, основными из которых являются `<title>...</title>` и `<meta>...</meta>`.

Тег `<title>` содержит информацию, которая будет выводиться в заголовочной части окна браузера пользователя. Например, если в этом теге имеется текст

```
<title>Мир книг</title>
```

то он отобразится в окне браузера так, как показано на рис. 2.4.

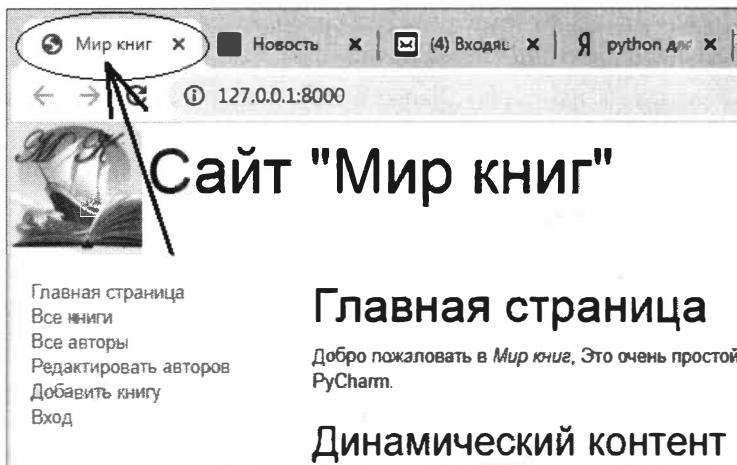


Рис. 2.4. Вывод содержимого тега `<title>` в браузере пользователя

Тег `<meta>` содержит специальную информацию:

- тип кодировки:

```
<meta charset="utf-8" />
```

- список ключевых слов:

```
<meta name="keywords" content="СУБД, БД, Базы данных">
```

В первом случае этот тег обеспечивает поддержку необходимой кодировки, а во втором — позволяет поисковым машинам корректно индексировать страницы сайта по ключевым словам. Следует заметить, что современные поисковые системы могут игнорировать ключевые слова, но это не отменяет возможность использования данного атрибута.

В рассмотренных тегах фрагменты `name="keywords"` и `content="список ключевых слов"` представляют собой *атрибуты* (параметры) тегов, которые конкретизируют их. Например, атрибуты могут указывать, что текст, заключенный в том или ином теге, при отображении должен выравниваться по центру. Атрибуты записываются сразу после имени тега, причем значения атрибутов заключаются в кавычки. Атрибутов у тега может быть несколько, но могут они и вовсе отсутствовать.

Приведем пример простейшей веб-страницы (листинг 2.2).

### Листинг 2.2. Простейшая веб-страница

```
<!DOCTYPE HTML>
<html>
    <head>
        <title>Веб-страница</title>
        <meta charset="windows-1251">
        <meta name="keywords" content="веб-страница первая">
    </head>
    <body>
        Моя первая веб-страница
    </body>
</html>
```

Текст страницы можно набрать в любом редакторе и сохранить в файле с расширением `htm` или `html`. Такой файл можно открыть при помощи какого-либо браузера.

## 2.2.1. Теги для представления текста на HTML-страницах

Текст, содержащийся в теле документа, обычно обрамляется специальными тегами. Наиболее распространены следующие виды тегов.

Тег `<hi>` — для вывода заголовков, где *i* имеет значения от 1 до 6. Например:

```
<h1> ... </h1>
<h2> ... </h2>
<h3> ... </h3>
<h4> ... </h4>
<h5> ... </h5>
<h6> ... </h6>
```

Заголовки отображаются полужирным шрифтом. При этом у текста заголовка в теге `<h1>` наибольший размер шрифта, а у тега `<h6>` — наименьший. К этому тегу может быть добавлен параметр `align`, который определяет правило выравнивания заголовка относительно одной из сторон документа. Этот параметр может принимать следующие значения:

- `left` — выравнивание по левому краю;
- `center` — по центру;
- `right` — по правому краю;
- `justify` — выравнивание по ширине.

Тег `<p>` — для вывода параграфов (абзацев). К этому тегу также может быть добавлен параметр `align`. Каждый новый тег `<p>` с этим параметром устанавливает выравнивание параграфа относительно одной из сторон документа.

Тег `<br>` — служит для перевода строки.

Тег **<HR>** — предназначен для вывода горизонтальной линии. Параметр align этого тега определяет выравнивание линии относительно одной из сторон документа. Этот тег имеет еще несколько параметров:

- size — высота линии;
- width — ширина линии;
- color — цвет линии;
- noshade — отключает отбрасывание тени.

Представленные далее теги меняют следующие параметры текста:

- <B>** — полужирный текст;
- <I>** — наклонный текст;
- <U>** — подчеркнутый текст;
- <TT>** — моноширинный текст.

Тег **<PRE>** — выводит заранее отформатированный текст. Используется он для того, чтобы текст с пробелами, символами перехода на новые строки и символами табуляции корректно отображался браузером. В секциях **<PRE>** могут присутствовать гипертекстовые ссылки, однако применять другие HTML-теги нельзя.

Тег **<FONT>** — задает некоторые свойства шрифта. Эти свойства определяются следующими параметрами:

- size — размер шрифта от 1 до 7;
- face — начертание шрифта;
- color — цвет шрифта.

В листинге 2.3 приведен пример использования некоторых тегов.

### Листинг 2.3. Пример использования тегов

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset=windows-1251">
    <title>Моя первая веб-страница</title>
  </head>
  <body>
    <p>Моя первая веб-страница</p>
    <h1>Заголовок h1</h1>
    <h2>Заголовок h2</h2>
    <h3>Заголовок h3</h3>
    <h4>Заголовок h4</h4>
    <h5>Заголовок h5</h5>
    <h6>Заголовок h6</h6>
  </body>
</html>
```

Открывающие теги могут содержать дополнительную информацию в виде параметров (атрибутов), которые существенно расширяют возможности представления информа-

ции. Параметры в открывающем теге записываются после названия тега в виде параметр="значение" и разделяются пробелами. Порядок следования параметров в теге непроизвольный. Если параметр отсутствует, его значение принимается по умолчанию согласно спецификации.

Вот основные параметры тега <BODY>:

- text — устанавливает цвет текста документа; значение цвета представлено в виде RRGGBB (например, text="000000" — черный цвет);
- link — устанавливает цвет гиперссылок, значение цвета имеет вид RRGGBB (например, text="FF0000" — красный цвет);
- vlink — устанавливает цвет гиперссылок, на которых пользователь уже побывал (например, text="00FF00" — зеленый цвет);
- alink — устанавливает цвет гиперссылок при нажатии на них (например, text="0000FF" — синий цвет);
- bgcolor — устанавливает цвет фона документа; значение цвета представлено в виде RRGGBB (например, text="FFFFFF" — белый цвет);
- background — устанавливает изображение для фона документа (например, background="bg.gif");
- topmargin — задает величину верхнего поля документа (например, topmargin="0");
- leftmargin — задает величину левого поля документа (например, leftmargin="10").

#### **ПРИМЕЧАНИЕ**

В HTML-коде цвет определяется 6-значным кодом RRGGBB (красный, красный, зеленый, зеленый, синий, синий), а в JavaScript или в CSS таким же 6-значным кодом или английским названием цвета. Со значениями кодов, обозначающими цвета, можно ознакомиться по следующей ссылке: <https://www.rapidtables.com/web/color/html-color-codes.html>.

Вот пример использования в теге <BODY> указанных параметров:

```
<BODY text="black" bgcolor="white">
```

Несмотря на то что описанные здесь параметры форматирования еще весьма широко распространены, нужно воздерживаться от их применения, т. к. для этих целей предназначены средства каскадных таблиц стилей, о чем речь пойдет в последующих разделах.

### **2.2.2. Списки**

Современным стандартом HTML предусмотрены три основных вида списков:

- маркированные списки (unordered list);
- нумерованные списки (ordered list);
- списки определений (definition list).

*Маркированные списки* названы так потому, что перед каждым пунктом таких списков устанавливается тот или иной *маркер*. Иногда, прибегая к дословному переводу, их также называют *неупорядоченными списками* (unordered list). Маркированные списки

задаются при помощи тегов `<ul>...</ul>`. Для задания элементов списка (item list) используются теги `<li>...</li>`. Например:

```
<ul>
<li>Пункт 1</li>
<li>Пункт 2</li>
<li>Пункт 3</li>
</ul>
```

Помимо элементов списка внутри тегов `<ul>...<ul>` можно размещать и другие теги — например, теги заголовков:

```
<ul>
<h3>Маркированный список</h3>
<li>Пункт 1</li>
<li>Пункт 2</li>
<li>Пункт 3</li>
</ul>
```

Тип маркера можно задавать с помощью атрибута `type`. Три его возможных значения:

- circle (незакрашенный кружок);
- disk (закрашенный кружок);
- square (закрашенный квадрат).

По умолчанию задан тип `disk`. Следует отметить, что различные модели браузеров могут по-разному отображать маркеры. Вот пример использования маркера типа `circle`:

```
<ul type="circle">
<li>Пункт 1</li>
<li>Пункт 2</li>
<li>Пункт 3</li>
</ul>
```

В *нумерованных списках* (ordered list), которые иногда называют *упорядоченными*, каждому пункту присваивается номер. Создаются такие списки при помощи тегов `<ol>...</ol>`. Для элементов нумерованных списков, как и в случае маркированных списков, также используются теги `<li>...</li>`. В таких списках, как и в маркированных, доступны пять типов маркеров, определяемых при помощи атрибута `type`, который может принимать следующие значения:

- 1 — арабские цифры;
- i — строчные римские цифры;
- I — прописные римские цифры;
- a — строчные латинские буквы;
- A — прописные латинские буквы.

Далее приведены примеры нумерованных списков:

```
<ol>
<h3>Нумерованный список</h3>
<li>Пункт 1</li>
<li>Пункт 2</li>
<li>Пункт 3</li>
</ol>
```

```
<ol type="I">
<li>Пункт 1</li>
<li>Пункт 2</li>
<li>Пункт 3</li>
</ol>
```

*Списки определений* (definition list) предназначены для того, чтобы организовать текст по примеру словарных статей. Они задаются с помощью тегов `<dl>...</dl>`, а определяемый термин или понятие (definition term) помещается в теги `<dt>...</dt>`. Определение понятия (definition description) заключается в теги `<dd>...</dd>`. В тексте, содержащемся внутри тегов `<dt>...</dt>`, недопустимы теги уровня блока, такие как `<p>` или `<div>`. Как и в предыдущих случаях, внутри списков определений могут быть теги заголовков и прочие теги:

```
<dl>
<h3>Список определений</h3>
<dt>Понятие 1</dt>
<dd>Определение понятия 1</dd>
<dt>Понятие 2</dt>
<dd>Определение понятия 2</dd>
</dl>
```

Как маркированные, так и нумерованные списки можно вкладывать друг в друга, причем допускается произвольное вложение различных типов списков. При вложении друг в друга маркированных и нумерованных списков следует быть внимательным. В листинге 2.4 приведен пример вложенных списков.

#### Листинг 2.4. Пример вложенных списков

```
<ul>
<h3>Пример вложенных списков</h3>
<li>Пункт 1</li>
<ol>
<li>Пункт 1.1</li>
<li>Пункт 1.2</li>
</ol>
<li>Пункт 2</li>
<ol type="i">
<li>Пункт 2.1</li>
<li>Пункт 2.2</li>
<li>Пункт 2.3</li>
</ol>
<li>Пункт 3</li>
<ol type="I">
<li>Пункт 3.1</li>
</ol>
</ul>
```

### 2.2.3. Таблицы

Таблицы — один из основных элементов, предназначенных для структурирования информации в HTML-документах. Хотя сейчас такое использование таблиц признано

устаревшим и нерекомендуемым, его до сих пор применяют многие веб-дизайнеры. Таблица создается при помощи тега <TABLE>, который может иметь следующие параметры:

- align — задает выравнивание таблицы (align=left/center/right);
- border — задает толщину линий таблицы (в пикселях);
- bgcolor — устанавливает цвет фона документа;
- background — устанавливает изображение фона документа;
- cellpadding — задает ширину промежутков между содержимым ячейки и ее границами (в пикселях), т. е. определяет поля внутри ячейки;
- cellspacing — задает ширину промежутков между ячейками (в пикселях);
- width — задает ширину таблицы в пикселях, процентах или частях.

Для создания заголовка таблицы служит тег <CAPTION>. По умолчанию заголовки центрируются и размещаются либо над (<CAPTION align="top">), либо под таблицей (<CAPTION align="bottom">). Заголовок может состоять из любого текста и изображений. Текст будет разбит на строки, соответствующие ширине таблицы. Каждая новая строка таблицы создается тегом <TR> (Table Row), который может иметь дополнительные параметры:

- align — задает горизонтальное выравнивание информации в ячейках (align=left/center/|right/justify);
- valign — задает вертикальное выравнивание информации в ячейках (valign=top/middle/bottom).

Внутри строки таблицы размещаются ячейки с данными, создаваемые тегами <TD>. Число тегов <TD> в строке определяет число ячеек (столбцов) таблицы. Тег <TD> может иметь следующие дополнительные параметры:

- align — задает горизонтальное выравнивание информации в ячейке (align=left/center/right/justify);
- valign — задает вертикальное выравнивание информации в ячейке (valign=top/middle/bottom);
- colspan — объединение столбцов в строке (например, colspan=2);
- rowspan — объединение строк в столбце (например, rowspan=3);
- width — задает ширину ячейки в пикселях или процентах;
- nowrap — запрещает переход текста в ячейке на новую строку.

Для задания заголовков столбцов и строк таблицы служит тег заголовка <TH> (Table Header). Этот тег аналогичен тегу <TD> с той лишь разницей, что текст в теге <TH> по умолчанию выделяется полужирным шрифтом и располагается по центру ячейки.

В листинге 2.5 приведен пример простой таблицы.

#### Листинг 2.5. Пример таблицы

```
<table border="3" cellpadding="7" cellspacing="3" height="80" width="50%>
<caption>Пример простой таблицы</caption>
<tr align="center">
  <td>1.1</td>
```

```
<td>1.2</td>
<td>1.3</td>
</tr>
<tr align="center">
    <td align="center">2.1</td>
    <td align="right">2.2</td>
    <td>2.3</td>
</tr>
</table>
```

Благодаря наличию большого числа параметров, а также возможности создания границ нулевой толщины, таблица может выступать в роли невидимой модульной сетки, относительно которой добавляется текст, изображения и другие элементы, организуя информацию на странице.

- Можно отметить следующие возможности при табличной верстке страниц:
  - создание колонок;
  - создание «резинового» макета;
  - «склейка» изображений;
  - включение фоновых рисунков;
  - выравнивание элементов.
- При этом табличная верстка имеет и ряд недостатков:
  - долгая загрузка;
  - громоздкий код;
  - плохая индексация поисковиками;
  - нет разделения содержимого и оформления;
  - несоответствие стандартам.

Рассмотрим некоторые типовые модульные сетки.

**Двухколонная модульная сетка** часто применяется на небольших информационных сайтах. Как правило, в первой колонке располагается логотип и меню сайта, а во второй — основной материал. Пример такой модульной сетки приведен на рис. 2.5.

В листинге 2.6 приведен HTML-код, реализующий эту структуру.

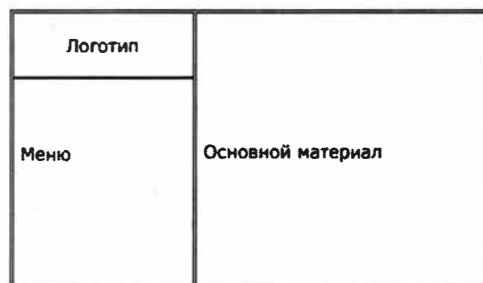


Рис. 2.5. Двухколонная модульная сетка

### Листинг 2.6. Пример HTML-кода двухколонной модульной сетки

```
<TABLE width="760" cellpadding="10" cellspacing="10" border="1">
<TR>
    <TD align="center">Логотип</TD>
    <TD rowspan="2">Основной материал</TD>
</TR>
<TR>
    <TD>Меню</TD>
</TR>
</TABLE>
```

**Трехколонная модульная сетка** применяется на крупных порталах с множеством информационных сервисов. Как правило, в первой колонке располагается меню сайта и дополнительная информация, во второй — основной материал, в третьей — дополнительные функции. Часто в модульную сетку сайта добавляют блоки «Заголовок сайта» и «Окончание сайта» (этот блок еще называют «подвалом»). Пример такой модульной сетки приведен на рис. 2.6.

В листинге 2.7 приведен HTML-код, реализующий эту структуру.

### Листинг 2.7. Пример HTML-кода трехколонной модульной сетки

```
<TABLE width="100%" cellpadding="10" cellspacing="10" border="0">
<TR align="center">
    <TD colspan="3">Заголовок сайта</TD>
</TR>
<TR>
    <TD width="200px">Меню</TD>
    <TD width="*">Основной материал</TD>
    <TD width="200px">Дополнительные функции</TD>
</TR>
<TR align="center">
    <TD colspan="3">Окончание сайта</TD>
</TR>
</TABLE>
```



Рис. 2.6. Трехколонная модульная сетка

## 2.2.4. Тег *div*

Буквально 2–3 года назад «скелетом» для сайта служила таблица (тег `<table>`). Создавалась одна большая таблица, которая потом делилась на несколько областей: заголовок, левый блок, правый блок, центр и низ. С появлением тега `<div>` все веб-мастера стали пользоваться им для создания «скелета» будущего сайта.

Тег разделения HTML-страниц, сокращенно называемый `<div>`, представляет собой специальный элемент, который позволяет объединять похожие наборы содержимого веб-страницы в условные группы. Элемент `<div>` появился в стандарте HTML 3.0 и представлял собой «участок» контента. Его можно использовать как универсальный контейнер для связывания похожего содержимого. В первую очередь тег `div` применяется для группировки похожего содержимого, чтобы его можно было легко стилизовать. Отличный пример — группировка различных разделов веб-страницы с помощью `<div>`. Например, можно объединить верхнюю навигационную панель и нижний колонтитул страницы в отдельный тег `div`, чтобы эти разделы можно было стилизовать вместе. Вот синтаксис тега `div`:

```
<div class="Имя класса">  
...  
</div>
```

Этот тег имеет ряд атрибутов:

- `— задает выравнивание;`
- `— определяет принадлежность к классу;`
- `— задает стили;`
- `— задает всплывающая подсказка к тегу.`

Атрибут  `(выравнивание) может принимать следующие значения:`

- `— выравнивание текста по центру;`
- `— выравнивание текста по левому краю;`
- `— выравнивание текста по правому краю;`
- `— выравнивание по ширине.`

В листинге 2.8 приведен пример использования тега `<div>`.

### Листинг 2.8. Пример использования тега `<div>`

```
<html>  
<head>  
<style type="text/css">  
    .center {  
        text-align: center;  
    }  
</style>  
</head>  
<body>  
    <div class="center">  
        Этот текст будет выровнен по центру  
    </div>
```

```
</body>
</html>
```

В данном примере текст, заключенный в тег `<div>`, будет выровнен по центру страницы. Более детально использование этого тега будет показано на примерах в разделе, посвященном фреймворку Bootstrap.

## 2.2.5. Гиперссылки

Для связи различных документов HTML предусматривает использование *ссылок*. Сам термин HTML (Hyper Text Markup Language) подразумевает их широкое применение. Для реализации ссылок в HTML служит тег `<a>...</a>`, который, как и большинство HTML-тегов, является контейнерным. Основной атрибут этого тега — `href`, собственно и содержащий адрес ресурса, на который указывает ссылка. Внутри тега `<a>...</a>` помещается текст ссылки.

В ссылках возможна как относительная, так и абсолютная адресация. При абсолютной адресации атрибуту `href` присваивается абсолютный URL-адрес ресурса:

```
<a href="http://server.com/doc3.htm">Ссылка на документ с абсолютным адресом  
http://server.com/doc3.htm</a>
```

При относительной адресации указывается путь к документу относительно текущей страницы:

```
<a href="doc1.htm">Ссылка на документ с относительным адресом doc1.htm</a>
```

Если в заголовочной части документа указан тег `<base>`, то отсчет будет вестись от адреса, заключенного в этом теге.

Помимо веб-страниц, допускается ссылаться и на другие интернет-ресурсы: e-mail, ftp, Gopher, WAIS, Telnet, newsgroup. Далее приведен пример ссылки на адрес электронной почты:

```
<a href="mailto:sss@mail.ru">Ссылка на адрес электронной почты sss@mail.ru</a>
```

В роли ссылок могут выступать и рисунки. Для этого сначала нужно вставить рисунок с помощью тега `<img>`. У атрибута `src` этого тега устанавливается значение, соответствующее имени файла рисунка:

```

```

Далее, рисунок "обертывается" в тег ссылки:

```
<a href="link2.htm"></a>
```

## 2.3. Каскадные таблицы стилей (CSS)

Если HTML служит для логического форматирования документа, то для управления его отображением на экране монитора или выводом на принтер применяются каскадные таблицы стилей (CSS). Технология CSS реализует концепцию «Документ — представление» и позволяет отделять оформление HTML-документа от его структуры. Кроме того, CSS существенно расширяет возможности представления (оформления) документов, внося множество новых возможностей.

Для того чтобы таблица стилей влияла на вид HTML-документа, она должна быть подключена к нему. Подключить каскадные таблицы стилей можно с использованием внешних, внутренних или локальных таблиц стилей:

- внешние таблицы стилей, оформленные в виде отдельных файлов, подключаются к HTML-документу при помощи тега в заголовке документа. Например:

```
<LINK rel="stylesheet" href="style.css" type="text/css">
```

- внутренние таблицы стилей в составе HTML-документа помещаются в заголовок страницы при помощи тега <STYLE>. Например:

```
<HEAD>
<STYLE>
    P {color: #FF0000}
</STYLE>
</HEAD>
```

- локальные таблицы стилей объявляются непосредственно внутри тега, к которому они относятся, при помощи параметра style. Например:

```
<P style="color: #FF0000">Каскадные таблицы стилей</P>.
```

Таблицы стилей записываются в виде последовательности тегов, для каждого из которых в фигурных скобках указываются его свойства по схеме параметр: свойство. Параметры внутри фигурных скобок разделяются точкой с запятой. Например:

```
P {color: #CCCCCC; font-size: 10px}
H1{color: #FF0000}
```

С помощью каскадных таблиц стилей можно менять свойства следующих элементов: фона и цвета, шрифта, текста, полей и отступов, границ.

Для задания свойств фона и цвета служат следующие параметры:

- background-color — цвет заднего плана;
- background-image — изображение заднего плана;
- background-repeat — дублирование изображения заднего плана (значения: repeat/repeat-x/repeat-y/no-repeat);
- background-attachment — фиксация изображения заднего плана (значения: scroll/fixed);
- background-position — начальное положение изображения заднего плана.

Для задания свойств шрифта служат следующие параметры:

- font-style — стиль шрифта (значения: normal / italic);
- font-weight — начертание шрифта (значения: normal / bold);
- font-size — размер шрифта;
- font-family — список имен шрифтов в порядке их приоритета.

Для задания свойств текста служат следующие параметры:

- word-spacing — установка промежутка между словами;
- letter-spacing — установка высоты строки;

- text-indent — установка абзацного отступа;
- text-align — выравнивание текста;
- vertical-align — установка вертикального выравнивания текста;
- text-decoration — преобразование текста (значение: none/underline/overline/line-through/blink).

Для задания свойств полей и отступов служат следующие параметры (`margin` — отступы между элементами, `padding` — отступ от содержимого элемента до его границы):

- margin-top — установка верхнего поля;
- margin-right — установка правого поля;
- margin-bottom — установка нижнего поля;
- margin-left — установка левого поля;
- margin — установка всех полей (например: `margin: 10px 10px 10px 0px`);
- padding-top — установка верхнего отступа;
- padding-right — установка правого отступа;
- padding-bottom — установка нижнего отступа;
- padding-left — установка левого отступа;
- padding — установка всех отступов (например: `padding: 10px 10px 10px 0px`).

Для задания свойств границ служат следующие параметры:

- border-width — установка ширины границы;
- border-color — установка цвета границы;
- border-style — установка стиля границы (значения: none/dotted/dashed/solid/double).

Для придания управлению элементами HTML большей гибкости используются *классы*, которые позволяют задавать различные стили для одного и того же тега. В таблицах стилей имя класса записывается после тега и отделяется от него точкой. Например:

```
P.green {color: #00FF00}  
P.blue {color: #0000FF}
```

В HTML-коде соответствие тега определенному классу указывается при помощи параметра `class`. Например:

```
<p class="green">Зеленый текст</p>  
<p class="blue">Синий текст</p>
```

## 2.4. Возможности использования JavaScript

Чаще всего код на языке JavaScript встраивается в веб-страницы для получения программного доступа к их элементам. Сценарии JavaScript являются основой клиентской части веб-приложений. Они загружаются с сервера вместе с веб-страницами и выполняются браузером на компьютере пользователя. Обработка сценариев JavaScript осуществляется встроенным в браузер интерпретатором. Что может делать JavaScript?

- В первую очередь JavaScript умеет отслеживать действия пользователя (например, реагировать на щелчок мыши или нажатие клавиши на клавиатуре, на перемещение курсора, на скроллинг).
- Менять стили, добавлять (удалять) HTML-теги, скрывать (показывать) элементы.
- Посыпать запросы на сервер, а также загружать данные без перезагрузки страницы (эта технология называется AJAX).

JavaScript — это объектно-ориентированный язык. Он поддерживает несколько встроенных объектов, а также позволяет создавать или удалять свои собственные (пользовательские) объекты. Объекты JavaScript могут наследовать свойства непосредственно друг от друга, образуя цепочку «объект -> прототип».

Сценарии JavaScript бывают встроенные, т. е. их содержимое является частью документа, и внешние, хранящиеся в отдельном файле с расширением js. Сценарии можно внедрить в HTML-документ следующими способами:

- в виде гиперссылки;
- в виде обработчика события;
- внутрь элемента <script>.

Рассмотрим эти способы подробнее.

Если подключать JavaScript в виде гиперссылки, то для этого код скрипта нужно разместить в отдельном файле, а ссылку на файл включить либо в заголовок:

```
<head>
  <script src="script.js"></script>
</head>
```

либо в тело страницы:

```
<body>
  <script src="script.js"></script>
</body>
```

Этот способ обычно применяется для сценариев большого размера или сценариев, многократно используемых на разных веб-страницах.

Можно подключать JavaScript к обработчику события. Дело в том, что каждый HTML-элемент может иметь JavaScript-события, которые срабатывают в определенный момент. Нужно добавить необходимое событие в HTML-элемент как атрибут, а в качестве значения этого атрибута указать требуемую функцию. Функция, вызываемая в ответ на срабатывание события, и станет обработчиком события. В результате срабатывания события исполнится связанный с ним код. Этот способ применяется в основном для коротких сценариев — например, можно установить смену цвета фона при нажатии на кнопку (листинг 2.9).

#### Листинг 2.9. Пример обработки события

```
<script>
var colorArray = ["#5A9C6E", "#A8BF5A", "#FAC46E", "#FAD5BB",
                  "#F2FEFF"]; // создаем массив с цветами фона
var i = 0;
```

```
function changeColor(){
    document.body.style.background = colorArray[i];
    i++;
    if( i > colorArray.length - 1){
        i = 0;
    }
}
</script>
<button onclick="changeColor()">Сменить цвет фона</button>
```

Код внутри элемента `<script>` можно вставлять в любое место документа. Код, который выполняется сразу после прочтения браузером или содержит описание функции, которая выполняется в момент ее вызова, располагается внутри тега. Описание функции можно располагать в любом месте — главное, чтобы к моменту ее вызова код функции уже был загружен.

Обычно код JavaScript размещается в заголовке документа (в элементе `<head>`) или после открывающего тега `<body>`. Если скрипт используется после загрузки страницы (например, код счетчика), то его лучше разместить в конце документа:

```
<footer>
    <script>
        document.write("Введите свое имя");
    </script>
</footer>
</body>
```

В примерах этой книги мы не будем писать собственные скрипты JavaScript, поскольку подключим готовые скрипты фреймворка Bootstrap, однако полезно знать о наличии этого языка и его возможностях при создании веб-страниц.

## 2.5. Краткие итоги

В этой главе мы познакомились с базовыми сведениями о веб-программировании и основными языками, с помощью которых можно разрабатывать клиентскую часть веб-приложения. Процедуры создания клиентской части веб-приложений значительно упрощаются, если использовать специализированные фреймворки для фронтенд-программирования. Наиболее популярным среди программистов является фреймворк Bootstrap, с которым мы более детально познакомимся в следующей главе.



## ГЛАВА 3

# Макетирование HTML-страниц с фреймворком Bootstrap

Фреймворк Bootstrap позволяет ускорить процесс верстки HTML-страниц и сделать готовый продукт адаптивным к разным вариантам разрешения и формата экрана. Он распространяется бесплатно в виде файлов формата HTML, CSS и JS. Это один из самых популярных фреймворков в мире. Основная область применения — разработка интерфейса сайтов, веб-приложений и среды администрирования. Любому веб-разработчику необходимо знать хотя бы базовые сведения об этом фреймворке и использовать его в своей работе.

Из материалов данной главы вы узнаете:

- о технологиях, заложенных в основу Bootstrap;
- как можно получить файлы фреймворка Bootstrap;
- что такое контейнеры и сетка Bootstrap;
- как можно сверстать макет HTML-страниц;
- как подключить файлы фреймворка Bootstrap к проекту;
- как задать цвет элементам HTML-страниц;
- как задать отступы в элементах макета HTML-страниц;
- как выровнять содержимое в адаптивных блоках HTML-страниц;
- как можно обозначить границы элементов в макете HTML-страниц;
- как использовать адаптивные контейнеры при макетировании HTML-страниц.

### 3.1. Технологические возможности фреймворка Bootstrap

Bootstrap — это свободный набор инструментов для создания сайтов и веб-приложений. Он включает в себя CSS-шаблоны и JavaScript, которые можно использовать для оформления кнопок, меток, блоков навигации и прочих компонентов веб-интерфейса. Фреймворк Bootstrap применяют по всему миру не только независимые разработчики, но и целые компании.

Современные веб-приложения адаптированы под различные экраны: вся информация должна помещаться в поле экрана без горизонтальной полосы прокрутки. Bootstrap по-

зволяет создать привлекательный дизайн интерфейса приложения и абсолютную адаптивность под разрешения различных экранов.

Bootstrap очень популярен по всему миру, т. к. позволяет верстать сайты в несколько раз быстрее, чем это можно выполнить на «чистом» CSS и JavaScript. Кроме того, популярность этого фреймворка обусловлена доступностью и легкостью использования. Его можно свободно подключить к своему приложению, и даже начинающий разработчик сможет верстать достаточно качественные макеты сайтов, которые трудно было бы выполнить без глубоких знаний веб-технологий и практических навыков.

Bootstrap был создан в Twitter в середине 2010 года и изначально предназначался исключительно для внутренних нужд компании. Однако уже через год состоялся публичный выпуск Bootstrap как фреймворка с открытым исходным кодом. С тех пор были выпущены более двадцати релизов этого фреймворка. Уже в версии Bootstrap 2 была реализована адаптивная функциональность, а в версии Bootstrap 3 библиотека была адаптирована к мобильным устройствам. В Bootstrap 4 проект был серьезно модифицирован, чтобы учесть два ключевых архитектурных изменения: переход на новые таблицы стилей — Sass (Syntactically Awesome Stylesheets) и на гибкие макеты страниц — CSS Flexbox.

**Sass** — это скриптовый метаязык (т. е. язык, описывающий другой язык), разработанный для упрощения файлов CSS. Синтаксис Sass полностью совместим с синтаксисом CSS, но при этом он короче, в нем отсутствуют скобки и точки с запятой, поэтому набирать код проще. Отступы имеют логическое значение, поэтому крайне важно следить за ними, т. к. неправильный отступ может сломать таблицу стилей. Такой синтаксис определенно понравится тем, кто программирует на Python. Sass дает возможность использовать переменные, вложенные правила, миксины, инлайновые импорты и многое другое, что совместимо с синтаксисом CSS. Sass упрощает создание больших таблиц стилей, а небольшим таблицам стилей позволяет работать быстро.

**CSS Flexbox** — это технология для создания сложных гибких макетов за счет правильного размещения элементов на странице. С помощью данной технологии можно очень просто и гибко расставить элементы в контейнере, распределить доступное пространство между ними, и выровнять их тем или иным способом, даже если они не имеют конкретных размеров. Технология CSS Flexbox позволяет упростить создание адаптивного дизайна, методы позиционирования пригодны как для разметки целой страницы, так и ее отдельных блоков.

Последний выпуск Bootstrap 5 ориентирован на улучшение кодовой базы Bootstrap 4 с минимальным количеством серьезных критических изменений. Были улучшены существующие функции и компоненты, удалена поддержка старых браузеров и внедрены более перспективные технологии, такие как настраиваемые свойства CSS.

Применение Bootstrap имеет следующие преимущества:

1. Поддерживается всеми популярными браузерами.
2. Он легок в освоении. С помощью знаний основ HTML и CSS каждый начинающий разработчик может создавать качественные веб-приложения. Кроме того, на официальном сайте Bootstrap выложена хорошо структурированная документация.
3. Создает адаптивный дизайн. Приложения, созданные на Bootstrap, сами приспособливаются к десктопам, ноутбукам, планшетам и мобильным телефонам.

4. Он содержит красивые и функциональные встроенные компоненты, которые легко настраивать.
5. Это фреймворк с открытым исходным кодом и находится в свободном доступе.

Bootstrap состоит из следующих элементов:

- инструменты для создания макета приложения (оберточных контейнеров, мощной системы сеток, гибких медиаобъектов, адаптивных утилитных классов);
- классов для стилизации базового контента: текста, изображений и таблиц;
- готовых компонентов: кнопок, форм, горизонтальных и вертикальных навигационных панелей, слайдеров, выпадающих списков, аккордеонов, модальных окон, всплывающих подсказок и др.;
- утилитных классов для решения традиционных задач, наиболее часто возникающих перед веб-разработчиками: выравнивание текста, отображение и скрытие элементов, задание цвета, фона, отступов и т. д.

На текущий момент есть три версии, которые можно использовать в своих проектах:

- v3 — это 3.4.1;
- v4 — это 4.6.0;
- v5 — это 5.0.0.

Bootstrap 5 рекомендуется для проектов, которые предназначены только для современных браузеров (поддержка IE и других браузеров не нужна). В других случаях лучше подключать файлы Bootstrap 4. Третью версию в основном имеет смысл использовать, если нужна поддержка «старых» браузеров (IE8 и IE9).

Материалы данной книги опираются на последнюю версию фреймворка Bootstrap 5. При этом мы не будем разбирать все тонкости и элементы этого фреймворка, а остановимся только на ключевых моментах, касающихся разметки HTML-страниц, т. к. это нам пригодится при формировании шаблонов Django.

## 3.2. Получение файлов фреймворка Bootstrap

Есть несколько способов использовать фреймворк Bootstrap в своих приложениях:

- подключить свое приложение (сайт) к коду фреймворка через сеть доставки контента CDN (Content Delivery Networks);
- скачать готовые (скомпилированные) файлы и подключить их к своему приложению;
- скачать исходные файлы и скомпилировать (собрать) свой собственный пакет, который будет содержать только необходимые компоненты CSS и плагины JS.

Мы рассмотрим первые два варианта как наиболее простые и доступные.

**Использование сети CDN.** Наиболее простой способ — подключение приложения (сайта) к фреймворку через сеть доставки контента (CDN). CDN — это географически распределенная сетевая инфраструктура, обеспечивающая быструю доставку контента пользователям веб-сервисов и сайтов. Входящие в состав CDN серверы географически располагаются таким образом, чтобы сделать время ответа для пользователей сайта

(сервиса) минимальным. Для такого подключения Bootstrap страницы сайта в заголовке должны содержать ссылку на CSS- и JS-файлы. Далее приведена HTML-страница, которая содержит эти ссылки:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Bootstrap 5</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link ref="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/
        dist/css/bootstrap.min.css" rel="stylesheet">
    <script>
        src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/
            dist/js/bootstrap.bundle.min.js">
    </script>
</head>
<body>

</body>
</html>
```

Здесь в теге `<head>` заголовка сайта присутствует пять строчек. Первые две строки не имеют отношения в Bootstrap, но для начинающих изучать основы веб-программирования поясним их назначение. Тег `<title>` служит для вывода информации, которая появится на самой верхней вкладке интернет-браузера. Далее следуют две строки с метатегами, разберемся, что это за теги `<meta>` и для чего они нужны.

Тег `<meta>` (от англ. *meta information* — метаинформация) определяет данные, которые предназначены для браузеров и поисковых систем. Например, механизмы поисковых систем обращаются к метатегам для получения описания сайта, ключевых слов и других данных. Разрешается использовать более чем один метатег, все они размещаются в заголовке HTML-страницы — в теге `<head>`.

**Метатег** `<meta charset="UTF-8">` определяет кодировку символов, которая будет принята на сайте. В приведенном ранее коде тег означает, что на сайте будет использоваться 8-битовый формат преобразования Unicode, который является стандартной кодировкой символов в последней версии HTML 5. Эта строка должна быть включена в каждую созданную веб-страницу, т. к. она гарантирует правильное отображение символов любого языка в любом браузере. Кодировка UTF-8 гарантирует, что символы нелатинских языков не будутискажаться. Браузер Google Chrome автоматически установил кодировку UTF-8, поэтому вам не придется беспокоиться об этом при разработке дизайна для этого браузера. Но вам все равно нужно включать `<meta charset="UTF-8">` в каждый файл HTML на случай, если эта функция не поддерживается другими браузерами.

**Метатег** `<name="viewport">` определяет, как нужно настроить параметры окна браузера для конкретного устройства, особенно это актуально для мобильного телефона. В настоящее время важно, чтобы веб-страницы хорошо отображались на всех устройствах: на настольных компьютерах, ноутбуках, планшетах и особенно на мобильных телефонах. Для этого нужно включить метатег `<name="viewport">` в каждый файл HTML. Этот тег определяет, как страницы сайта будут отображаться на экранах разного размера и

сколько визуальной области браузера будет доступно пользователю. Каждое устройство имеет разные размеры и разрешение экрана. Как правило, у мобильных устройств видимая область экрана меньше, а у настольных компьютеров больше. Использование параметра `content="width=device-width"` — первый шаг к тому, чтобы страницы сайтов хорошо выглядели на мобильных устройствах. Это не позволяет страницам сайта, который просматривается на мобильном устройстве, выглядеть так, как если бы он отображался на ноутбуке, т. е. сильно уменьшенным с мелкими и плохо читаемыми символами. Это гарантирует, что HTML-страницы будут автоматически адаптированы под размеры экрана устройства. Параметр `initial-scale=1.0` устанавливает начальный масштаб при первой загрузке страницы браузером, т. е. разворачивает его на полный экран.

В дополнение к приведенным метатегам на странице можно разместить еще три полезных метатега.

**Метатег `<x-ua-compatible>`** отвечает за браузеры Microsoft IE8, IE9, IE10, IE11. Он говорит о том, что страница будет адаптирована к последней версии Internet Explorer. Всегда выбирайте последнюю версию Internet Explorer, т. е. `IE=edge`. Вот пример строки с данным метатегом:

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

**Метатег `<meta name="description">`**. Наличие этого метатега помогает поисковым системам ранжировать ваш сайт по сравнению с другими сайтами. Он предназначен в основном для целей SEO (поисковая оптимизация). Метатег описания позволяет кратко и лаконично объяснить, о чем ваш сайт, и может выглядеть примерно так:

```
<meta name="description" content="Разработка веб-приложений: Python, Django, Bootstrap, HTML. Программирование на Python.">
```

Что нужно учитывать при описании вашего сайта в этом метатеге:

- описание должно быть коротким, не превышающим 160 символов;
- в описании должны быть ключевые слова, по которым люди ищут услуги, предоставляемые вашим сайтом;
- четко укажите ваши преимущества перед конкурентами.

**Метатег `<meta name="author">`**. Еще один полезный метатег, который следует включить в описание сайта, — это имя автора:

```
<meta name="author" content="Семенов Алексей">
```

Эта информация показывает, кто является владельцем веб-сайта или кому принадлежат авторские права.

**Скачивание файлов Bootstrap.** Второй путь использования фреймворка Bootstrap — скачать готовые файлы и подключить их к своему приложению. Этот путь тоже не представляет особых трудностей. Скачать готовые файлы можно с официального сайта по следующей ссылке: <https://getbootstrap.com/docs/5.2/getting-started/download/>.

После скачивания вы получите архивированный zip-файл, в имени которого будет отражена текущая версия фреймворка, например `bootstrap-5.2.3-dist.zip`. После распаковки архива будут созданы две папки: `css` — с файлами стилей и `js` — с java-скриптами. Эти две папки нужно перенести в ваш проект.

**Подключение файлов Bootstrap.** Последние две строки обеспечивают подключение приложения к файлам CSS фреймворка Bootstrap через сеть доставки контента CDN:

```
<link href="https://cdn... — подключение приложения к файлам CSS;
```

```
<script src="https://cdn... — подключение приложения к файлам JS.
```

Вот, собственно, и все, в таком виде на HTML-странице можно использовать все доступные в Bootstrap классы, методы и свойства.

### 3.3. Контейнеры и сетка Bootstrap

Сначала разберемся с тем, что такое адаптивный макет сайта. Адаптивный макет — это такой макет, вид которого может изменяться в зависимости от того, какую ширину (viewport) имеет браузер. Это означает, что при одних значениях ширины viewport адаптивный макет может выглядеть одним образом, а при иных — совершенно по-другому. В Bootstrap изменение вида макета реализовано через медиазапросы. Каждый медиазапрос в Bootstrap строится на основании минимальной ширины браузера. В Bootstrap ключевое значение ширины viewport в медиазапросе называется *breakpoint* (контрольной точкой, или классом устройства). На рис. 3.1 приведены основные контрольные точки, которые в Bootstrap 4 и Bootstrap 5 назначены по умолчанию.

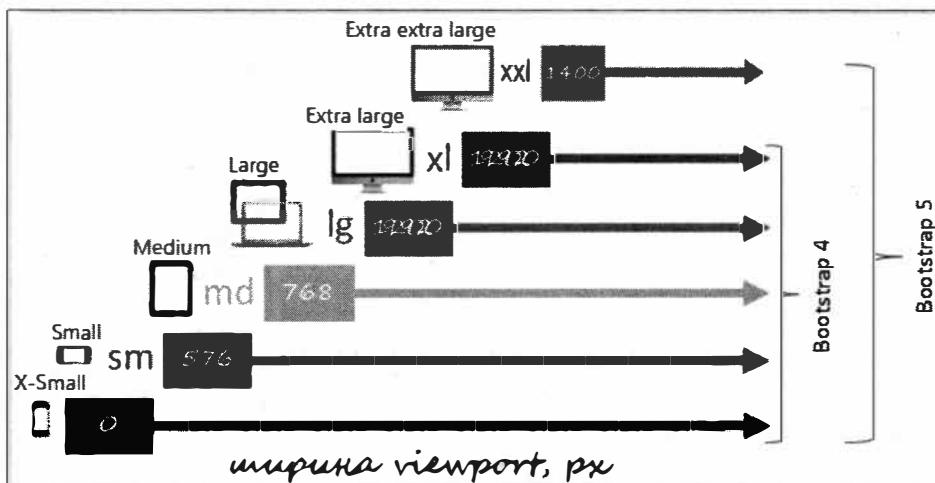


Рис. 3.1. Контрольные точки для Bootstrap 4 и Bootstrap 5

Цифры на этом рисунке означают, что при ширине видимой области браузера до 576 px макет сайта будет отображаться одним образом, от 576 до 768 px — другим образом и т. д. Иными словами, можно создать макет, который на каждом из этих участков видимой области браузера станет выглядеть по-разному.

Контрольные точки (breakpoint) имеют обозначение. Первая контрольная точка не имеет обозначения (т. е. `xs` не указывается), вторая обозначается — `sm`, третья — `md`, четвертая — `lg`, пятая — `xl` и шестая — `xxl`. Эти обозначения необходимо запомнить, т. к. они используются в классах, которые мы будем добавлять к элементам. Эти обозначения в имени класса будут указывать на то, с какой ширины `viewport` стили, определенные

в нем, будут применяться к вложенным элементам. При этом контрольные точки задают только минимальную ширину. Например, если вы определили макет, указав в нем классы без обозначения контрольной точки (`xs`) и с использованием параметра `md`, то на устройстве `sm` страница будет иметь такую же структуру, как и на устройстве `xs`, а на устройствах с большим экраном `lg`, `xl` и `x1l` — как на `md`.

Теперь разберемся с сеткой HTML-страницы в Bootstrap. Сетка состоит из следующих элементов:

- оберточных контейнеров (элементов с классом `container`, `container-fluid`, или `container-{breakpoint}`);
- рядов или строк (элементов с классом `row`);
- адаптивных колонок или блоков, имеющих один или несколько классов `col`.

Все части сетки — это обычные элементы HTML, к которым просто добавлены определенные классы.

### 3.3.1. Адаптивные контейнеры

Адаптивный (или оберточный) контейнер — это элемент сетки Bootstrap, с которого обычно начинается создание макета страницы или ее части. Другими словами, это базовый элемент, в котором необходимо размещать все другие элементы сетки (ряды и адаптивные блоки). Его основная цель — это установить шаблону ширину и выровнять его по центру страницы (рис. 3.2).



Рис. 3.2. Оберточный контейнер

Контейнеры в Bootstrap являются основным элементом макетирования страниц. Контейнеры служат для вкладывания в них других элементов интерфейса, формирования отступов и выравнивания или центрирования их содержимого. Хотя контейнеры могут быть вложенными, для большинства макетов вложенный контейнер не требуется.

Bootstrap поставляется с двумя базовыми контейнерами:

- адаптивно фиксированный контейнер (класс `container`) — устанавливает максимальную ширину (`max-width`) для каждой контрольной точки;
- адаптивно подвижный («резиновый») контейнер (класс `container-fluid`), который устанавливает максимальную ширину (`width: 100%`) во всех контрольных точках.

Для адаптивно фиксированного контейнера можно указать контрольную точку (`container-{breakpoint}`), при этом будет установлена максимальная ширина (`width: 100%`) до указанной контрольной точки.

Адаптивно-фиксированный контейнер предназначен для создания контейнера с постоянной шириной, которая будет оставаться постоянной только в пределах действия

определенной контрольной точки. Например, на контрольной точке sm до действия контрольной точки md он будет иметь одну фиксированную ширину, а на md до действия lg — другую фиксированную ширину. Единственная контрольная точка, на которой данный контейнер не будет иметь фиксированную ширину, — это breakpoint без обозначения. Здесь контейнер будет занимать 100%-ную ширину видимой области браузера. Пример расположения адаптивно-фиксированного контейнера в окне браузера представлен на рис. 3.3.



Рис. 3.3. Расположения адаптивно фиксированного контейнера в окне браузера

Адаптивно-фиксированный контейнер будет иметь следующие значения ширины:

- 100%-ную ширину при ширине viewport до 576 px;
- 540 px при ширине viewport от 576 до 768 px;
- 720 px при ширине viewport от 768 до 992 px и т. д.

В горизонтальном направлении контейнер располагается по центру окна браузера. Осуществляется это посредством установки ему CSS-свойств margin-left:auto и margin-right:auto в файле bootstrap.css. Создать адаптивно-фиксированный контейнер можно так:

```
<div class="container">...</div>
```

Второй вид контейнера — это адаптивно-подвижный, или «резиновый» контейнер. Он применяется тогда, когда необходимо создать полностью гибкий макет целой страницы или ее части. Данный контейнер на любых контрольных точках занимает полную (100%-ную) ширину видимой области браузера (рис. 3.4).



Рис. 3.4. Расположение адаптивно-подвижного контейнера в окне браузера

Создать адаптивно-подвижный контейнер можно так:

```
<div class="container-fluid">...</div>
```

Контейнеры классов `container` и `container-fluid` имеют внутренние отступы слева и справа по 15 px. Установка внутренних отступов контейнеров задается в файле `bootstrap.css` посредством CSS-свойств `padding-left:15px` и `padding-right:15px`. При верстке макета страницы не следует одни оберточные контейнеры помещать внутрь других.

### 3.3.2. Ряды или строки (row)

Ряд или строка — это элемент сетки страницы, который выступает в роли контейнера для адаптивных блоков (колонок). Это дочерний элемент контейнера и родительский элемент для адаптивных блоков (рис. 3.5).

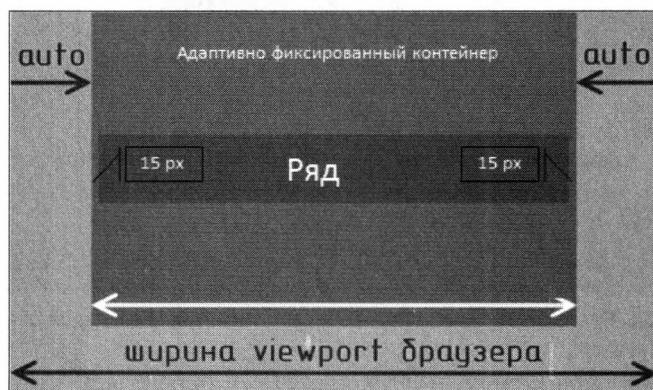


Рис. 3.5. Расположение ряда (строки) в теле адаптивно-фиксированного контейнера

Для того чтобы создать ряд, нужно использовать следующий код:

```
<div class="row">...</div>
```

В Bootstrap ряды играют очень важную роль. Это связано с тем, что сетка HTML-страниц построена на CSS Flexbox — адаптивных блоках. В этой сетке ряд выступает в роли flex-контейнера для flex-элементов (адаптивных блоков). Если адаптивные блоки окажутся вне ряда, то они работать не будут, в Bootstrap адаптивные блоки должны обязательно находиться в блоке с классом `row`. Компенсация внутренних отступов (padding) осуществляется за счет отрицательных левых и правых внешних отступов, равных 15 px (`margin-left:-15px` и `margin-right:-15px`).

### 3.3.3. Адаптивные блоки (col)

Адаптивные блоки — это основные строительные элементы сетки. Именно от них зависит то, как макет веб-страницы будет отображаться в видимой области браузера на разных контрольных точках. Адаптивные блоки располагаются внутри ряда, т. е. ряд является контейнером для адаптивных блоков (рис. 3.6).

Адаптивные блоки — это блоки, ширина которых на разных контрольных точках (breakpoint) может быть различной (в процентном отношении от родительского кон-

тейнера). Например, адаптивный блок в браузере смартфона (`sm`) может иметь ширину, равную 50% от родительского элемента, а на планшете (`md`) — 25%.

Адаптивный блок достаточно просто создать посредством добавления одного или нескольких классов `col-?-?` к HTML-элементу, расположенному в ряду. В классе `col-?-?` вместо первого знака вопроса указывается название контрольной точки («пусто», `sm`, `md`, `lg` или `xxl`). Вместо второго знака вопроса задается ширина адаптивного блока, которую он должен иметь на указанной контрольной точке. Ширина адаптивного блока назначается в относительной форме либо по умолчанию, либо с помощью числа от 1 до 12 (по количеству колонок).

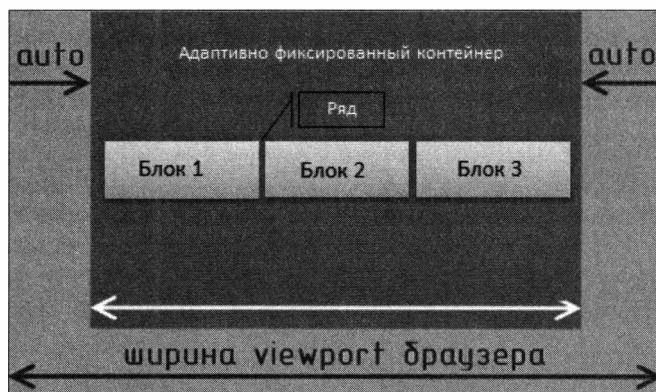


Рис. 3.6. Расположение адаптивных блоков в контейнере в ряд (в строке)

Число, которое стоит при инициализации класса `col`, определяет, какую часть ширины родительского контейнера (т. е. ряда) будет занимать адаптивный блок, начиная с указанной контрольной точки. При этом ширина ряда в числовом выражении (в колонках Bootstrap) по умолчанию равна 12. Например, блок с классом `col-md-4`, начиная с контрольной точки `md`, будет занимать  $4/12 \cdot 100\% = 33,3\%$ .

Адаптивные блоки, как и оберточные контейнеры, имеют внутренние отступы слева и справа по 15 px. Данные отступы адаптивным блокам во фреймворке Bootstrap устанавливаются с помощью CSS свойств `padding-left:15px` и `padding-right:15px`. Размещать адаптивные блоки необходимо в ряду (в строке), т. е. у любого адаптивного блока в качестве родителя должен быть обязательно элемент `row`.

Рассмотрим на примерах, какую ширину будут иметь следующие адаптивные блоки.

1. `class="col-12"` задает ширину блока по умолчанию. Ширина блока будет равна 12 колонкам Bootstrap (т. е.  $12/12 \cdot 100\% = 100\%$  от ширины ряда), эту ширину блок будет иметь, начиная с контрольной точки `xs`.
2. `class="col-sm-9"` задает ширину блока с контрольной точки `sm`. Ширина блока будет равна 9 колонкам Bootstrap (т. е.  $9/12 \cdot 100\% = 75\%$  от ширины ряда), эту ширину блок будет иметь, начиная с контрольной точки `sm`.
3. `class="col-md-7"` задает ширину блока с контрольной точки `md`. Ширина блока будет равна 7 колонкам Bootstrap (т. е.  $7/12 \cdot 100\% = 58,3\%$  от ширины ряда), начиная с контрольной точки `md`.

4. class="col-1g-5" задает ширину блока с контрольной точки lg. Ширина блока будет равна 5 колонкам Bootstrap (т. е.  $5/12 \cdot 100\% = 41,6\%$  от ширины ряда), начиная с контрольной точки lg.
5. class="col-xl-3" задает ширину блока с контрольной точки xl. Ширина блока будет равна 3 колонкам Bootstrap (т. е.  $3/12 \cdot 100\% = 25\%$  от ширины ряда), начиная с контрольной точки xl.
6. class="col-xxl-6" задает ширину блока с контрольной точки больше xxl. Ширина блока будет равна 2 колонкам Bootstrap (т. е.  $6/12 \cdot 100\% = 50\%$  от ширины ряда).

При задании ширины адаптивному блоку мы указываем класс, содержащий контрольную точку, начиная с которой данная ширина будет действовать. Эту ширину данный блок будет иметь до тех пор, пока она не будет переопределена с помощью другого класса, действие которого начинается с наибольшей ширины viewport.

### 3.3.4. Адаптивные блоки без указания числа колонок

В сетке Bootstrap имеются специальные классы col, col-sm, col-md, col-lg, col-xl, col-xxl, col-auto, col-sm-auto, col-md-auto, col-lg-auto, col-xl-auto и col-xxl-auto.

Первая группа классов (col, col-sm, col-md, col-lg, col-xl, col-xxl) предназначена для создания адаптивных блоков, ширина которых будет зависеть от свободного пространства в ряду (строке). Незанятая ширина (свободное пространство) в ряду между всеми такими блоками распределяется равномерно. Кроме того, данные адаптивные блоки перед распределением свободного пространства ряда (по умолчанию) имеют нулевую ширину.

### 3.3.5. Расположение адаптивных блоков

Адаптивные блоки по умолчанию располагаются в ряду горизонтальными линиями, при этом они выстраиваются последовательно слева направо. В одну горизонтальную линию могут поместиться адаптивные блоки с суммарным числом колонок не более 12.

Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12						
Col 6						Col 6											
Col 4				Col 4				Col 4									
Col 3		Col 3			Col 3			Col 3		Col 3							
Col 2		Col 2		Col 2		Col 2		Col 2		Col 2							
Col 8						Col 4											
Col 10										Col 2							
Col 4			Col 8														
Col 2		Col 10															

Рис. 3.7. Возможные варианты формирования адаптивных блоков из 12 колонок

Адаптивные блоки, которые не помещаются в текущую строку, переходят на следующую. Колонки можно объединять в адаптивные блоки произвольным образом. Пример различных вариантов расположения колонок в адаптивных блоках в ряду приведен на рис. 3.7.

## 3.4. Верстка макета HTML-страниц

Основной принцип верстки макета веб-страницы на сетке Bootstrap заключается во вкладывании адаптивных блоков в ряды, а рядов — в контейнеры. При этом ширина адаптивных блоков — это всегда относительная величина, которая задается в колонках Bootstrap и зависит только от ширины родителя, т. е. ряда. Размещать контент веб-страницы следует только в адаптивных блоках. Вот пример формирования макета страницы:

```
<div class="container">
  <div class="row">
    <div class=col-8>Блок 1</div>
    <div class=col-4>Блок 2</div>
  </div>
</div>
```

В этом примере был создан контейнер (`class="container"`), в него вложена строка (`class="row"`), и уже в эту строку вложено два адаптивных блока: шириной 8 колонок (`class=col-8`) и шириной 4 колонки (`class=col-4`).

Если предполагается, что блоки будут одинаковой ширины, то число колонок для них можно не указывать, например:

```
<div class="container">
  <div class="row">
    <div class="col"> Блок 1 </div>
    <div class="col"> Блок 2 </div>
    <div class="col"> Блок 3 </div>
    <div class="col"> Блок 4 </div>
    <div class="col"> Блок 5 </div>
  </div>
</div>
```

**Выравнивание адаптивных блоков** в горизонтальном и вертикальном направлениях осуществляется с помощью служебных flex-классов.

В пределах ряда по горизонтали адаптивные блоки выравнивают посредством одного из следующих классов, который необходимо дополнительно добавить к `row`:

- `align-items-start` — выравнивание относительно начала ряда (по левому краю);
- `align-items-center` — по центру ряда;
- `align-items-end` — относительно конца ряда (по правому краю).

Например, в следующем коде адаптивные блоки выровнены по центру ряда:

```
<div class="row align-items-center">
  <div class="col">Блок 1</div>
  <div class="col">Блок 2 </div>
</div>
```

По умолчанию адаптивные блоки занимают всю высоту ряда, в котором они расположены. Выравнивание какого-то определенного адаптивного блока по вертикали в пределах ряда может осуществляться одним из следующих классов:

- align-self-start — выравнивание относительно начала ряда (по верху ряда);
- align-self-center — по центру ряда;
- align-self-end — относительно конца ряда (по низу ряда).

Данные классы необходимо добавлять к адаптивным блокам, а не к ряду. Например, выравниваем адаптивный блок 2 по нижнему краю ряда:

```
<div class="row align-items-center">
    <div class="col">Блок 1</div>
    <div class="col align-self-end">Блок 2</div>
</div>
```

**Горизонтальное выравнивание адаптивных блоков.** Для выравнивания адаптивных блоков в горизонтальном направлении можно также использовать следующие классы:

- justify-content-start — относительно начала ряда (по левому краю), установлено по умолчанию;
- justify-content-center — по центру ряда;
- justify-content-end — относительно конца ряда (по правому краю);
- justify-content-around — равномерно, с учетом пространства перед первым и последним адаптивными блоками;
- justify-content-between — равномерно, с одинаковым пространством между адаптивными блоками.

Например, распределим адаптивные блоки в горизонтальном направлении равномерно:

```
<div class="row justify-content-around">
    <div class="col-4">Блок 1</div>
    <div class="col-4">Блок 2</div>
</div>
```

**Смещение адаптивных блоков** в Bootstrap можно выполнить с помощью следующих классов:

- классов offset — на определенное число колонок;
- служебных (утилитных) классов margin.

Классы offset предназначены для смещения адаптивных блоков вправо на определенное число колонок. Данные классы имеют следующий синтаксис:

- offset-{1};
- offset-{breakpoint}-{1}.

Здесь {breakpoint} — контрольная точка, начиная с которой к данному блоку будет применено смещение. Если она не указана, то смещение будет применено, начиная с самых крохотных устройств (смартфоны). Второй параметр {1} — величина смещения (число колонок). Рассмотрим следующий код:

```
<div class="row">
    <div class="col-4">Блок 1</div>
    <div class="col-4 offset-4">Блок 1</div>
</div>
<div class="row">
    <div class="col-3 offset-3">Блок 3 </div>
    <div class="col-3 offset-3">Блок 4</div>
</div>
<div class="row">
    <div class="col-6 offset-3">Блок 5</div>
</div>
```

В этом случае смещение адаптивных блоков будет выглядеть так, как показано на рис. 3.8.

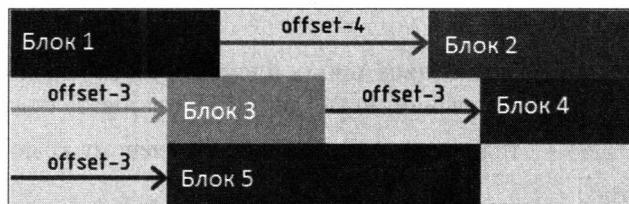


Рис. 3.8. Пример смещения адаптивных блоков с использованием класса offset

**Смещение с помощью классов margin.** В Bootstrap смещение адаптивным блокам также можно устанавливать с помощью отступов margin (margin-left:auto и margin-right auto). Этот вариант смещения появился благодаря тому, что сетка в новых версиях Bootstrap основывается на CSS Flexbox. Данный вариант удобен, когда блоки необходимо сместить относительно друг друга на некоторую переменную величину. Для более удобного задания блокам отступов margin можно указать сокращенное обозначение классов: ml-auto, mr-auto, ml-(breakpoint)-auto и mr-(breakpoint)-auto. Рассмотрим следующий код:

```
<div class="row">
    <div class="col-4">Блок 1</div>
    <div class="col-4 ml-auto">Блок 2</div>
</div>
<div class="row">
    <div class="col-3">Блок 3</div>
    <div class="col-3 ml-auto mr-auto">Блок 4</div>
    <div class="col-3">Блок 5</div>
</div>
<div class="row">
    <div class="col-4 ml-auto mr-auto">Блок 6</div>
    <div class="col-4 ml-auto mr-auto">Блок 7</div>
</div>
```

В этом случае смещение адаптивных блоков будет выглядеть так, как показано на рис. 3.9.

Итак, мы рассмотрели основные «строительные блоки», из которых создаются макеты HTML-страниц. Адаптивные блоки являются конечными элементами макета, в которые и размещается основной контент HTML-страниц.

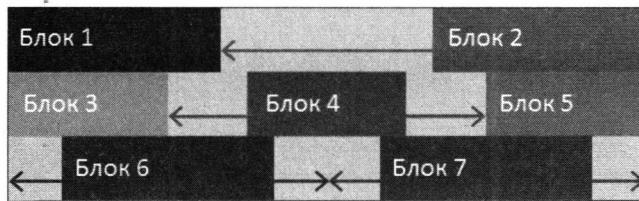


Рис. 3.9. Пример смещения адаптивных блоков с использованием класса margin

### 3.5. Подключение файлов фреймворка Bootstrap к проекту

Для изучения возможностей фреймворка Bootstrap откроем PyCharm и создадим в нем проект с именем Boot\_Start. В этом проекте создадим папку static и в нее перенесем две папки со скачанными файлами Bootstrap: css и js. Структура папок проекта приведена на рис. 3.10.

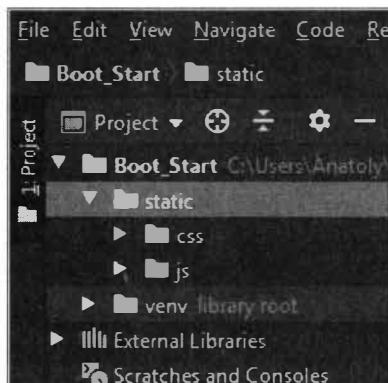


Рис. 3.10. Структура папок проекта Boot\_Start

Чтобы процесс создания макетов страниц был более понятен, разберем несколько простых примеров. Для начала создадим контейнер фиксированной ширины с одним рядом (строкой), в котором находится один адаптивный блок из 12 колонок, т. е. он будет занимать всю ширину ряда (листинг 3.1, страница start1.html).

#### Листинг 3.1. Страница start1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>start1</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="static/css/bootstrap.min.css" >
  <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
```

```
<body>
<div class="container">
    <div class="row">
        <div class="col">
            <h1>Это страница Bootstrap</h1>
            <p>На странице создан контейнер фиксированной ширины
                class="container".</p>
            <p>В контейнер вложена одна строка class="row."</p>
            <p>В строке находится один адаптивный блок class="col".</p>
            <p>С изменением размера окна браузера ширина элементов
                контейнера будет меняться в разных контрольных точках.</p>
        </div>
    </div>
</div>
</body>
</html>
```

При минимальном размере окна браузера данная страница будет выглядеть как на рис. 3.11.

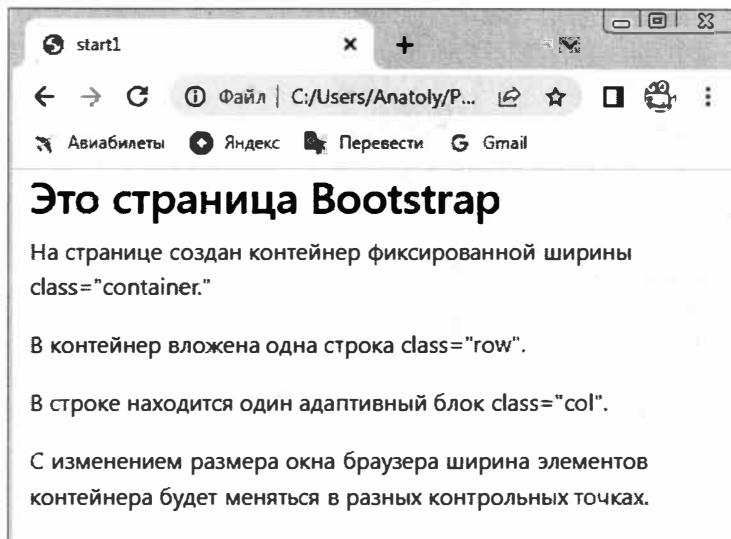


Рис. 3.11. Простейшая HTML-страница, созданная на основе Bootstrap

Как видно из данного рисунка, с параметрами по умолчанию текст на странице показан черным цветом на белом фоне, при этом отсутствуют какие-либо признаки, по которым можно определить размер и положение адаптивного блока, например рамки или выделение блока цветом. Для того чтобы наглядно продемонстрировать возможности макетирования страниц, необходимо явно показать границы блока. Это можно сделать либо с помощью цветного фона, либо созданием ограничительной рамки.

## 3.6. Задание цвета элементам HTML-страниц

Для задания цвета в Bootstrap предусмотрено несколько полезных классов. Класс `bg` позволяет задать 9 цветов:

- `class="bg-primary";`
- `class="bg-secondary";`
- `class="bg-success";`
- `class="bg-danger";`
- `class="bg-warning";`
- `class="bg-info";`
- `class="bg-light";`
- `class="bg-dark";`
- `class="bg-white".`

Пример HTML-кода страницы, в которой показано использование класса `bg` для задания цвета фону, приведен в листинге 3.2, страница `color_bg.html`.

### Листинг 3.2. Страница `color_bg.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>color_bg</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container">
    <div class="row">Первая строка</div>
    <div class="row">
        <div class="col bg-primary">Блок 1</div>
        <div class="col bg-secondary">Блок 2</div>
        <div class="col bg-success">Блок 3</div>
        <div class="col bg-danger">Блок 4</div>
        <div class="col bg-warning">Блок 5</div>
        <div class="col bg-info">Блок 6</div>
        <div class="col bg-light">Блок 7</div>
        <div class="col bg-dark">Блок 8</div>
        <div class="col bg-white">Блок 9</div>
    </div>
</div>
</body>
</html>
```

В окне браузера данная страница будет иметь вид, представленный на рис. 3.12.

Как видно из рис. 3.12, текст «Блок 8» не виден, т. к. и шрифт, и фон имеют одинаковый (черный) цвет.

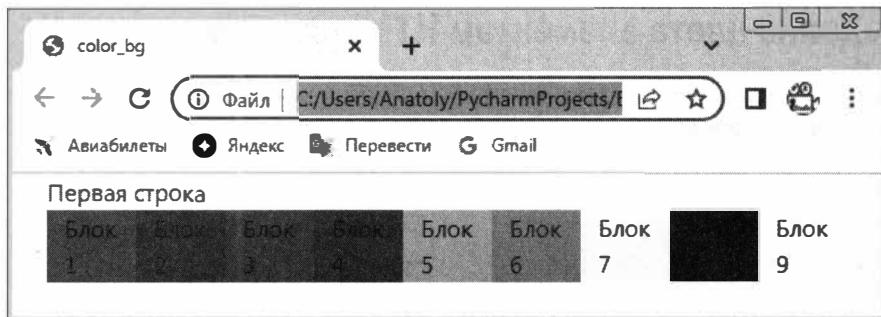


Рис. 3.12. Образцы цвета фона в Bootstrap

Класс `text` позволяет задать 10 цветов:

- `class="text-primary";`
- `class="text-secondary";`
- `class="text-success";`
- `class="text-danger";`
- `class="text-warning";`
- `class="text-info";`
- `class="text-light";`
- `class="text-dark";`
- `class="text-muted";`
- `class="text-white".`

Пример HTML-кода страницы, в которой показано использование класса `text` для задания цвета шрифту, приведен в листинге 3.3, страница `color-text.html`.

### Листинг 3.3. Страница `color-text.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>color|_ text</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container">
    <div class="row">Первая строка</div>
    <div class="row">
        <div class="col text-primary">Блок 1</div>
        <div class="col text-secondary">Блок 2</div>
        <div class="col text-success">Блок 3</div>
        <div class="col text-danger">Блок 4</div>
        <div class="col text-warning">Блок 5</div>
        <div class="col text-info">Блок 6</div>
        <div class="col text-light">Блок 7</div>
        <div class="col text-dark">Блок 8</div>
        <div class="col text-muted">Блок 9</div>
        <div class="col text-white">Блок 10</div>
    </div>
</div>
```

```
</div>
</body>
</html>
```

В окне браузера данная страница будет иметь вид, представленный на рис. 3.13.

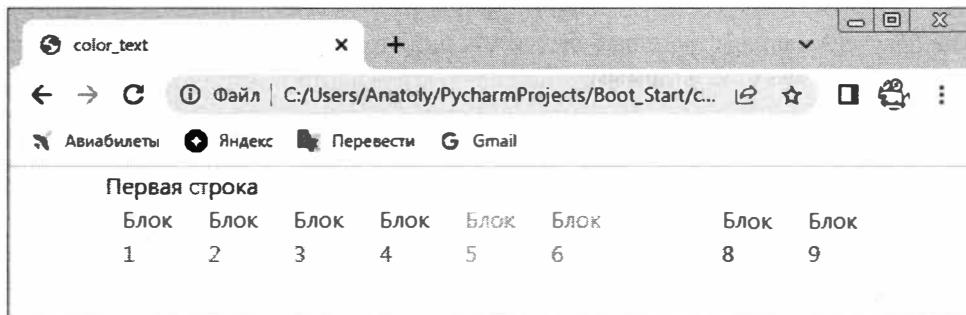


Рис. 3.13. Образцы цвета текста в Bootstrap

Как видно из данного рисунка, текст «Блок 7» и «Блок 10» не виден, т. к. и шрифт, и фон имеют одинаковый цвет.

Для того чтобы текст всегда был виден, он должен иметь цвет, отличный от цвета фона. Пример HTML-кода страницы, в которой показано использование двух классов `text` и `bg` для одновременного задания цвета для шрифта и для фона, приведен в листинге 3.4, страница `color_text_bg.html`.

#### Листинг 3.4. Страница `color_text_bg.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>color_text_bg</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="static/css/bootstrap.min.css" >
  <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
  <div class="container">
    <div class="row">Первая строка</div>
    <div class="row">
      <div class="col text-primary bg-info">Блок1</div>
      <div class="col text-secondary">Блок2</div>
      <div class="col text-success bg-info">Блок3</div>
      <div class="col text-danger">Блок4</div>
      <div class="col text-warning bg-info">Блок5</div>
      <div class="col text-info">Блок6</div>
      <div class="col text-light bg-primary">Блок7</div>
      <div class="col text-dark">Блок8</div>
```

```

<div class="col text-muted bg-info">Блок9</div>
<div class="col text-white bg-primary">Блок10</div>
</div>
</div>
</body>
</html>

```

В окне браузера данная страница будет иметь вид, представленный на рис. 3.14. Теперь текст во всех блоках стал видимым.

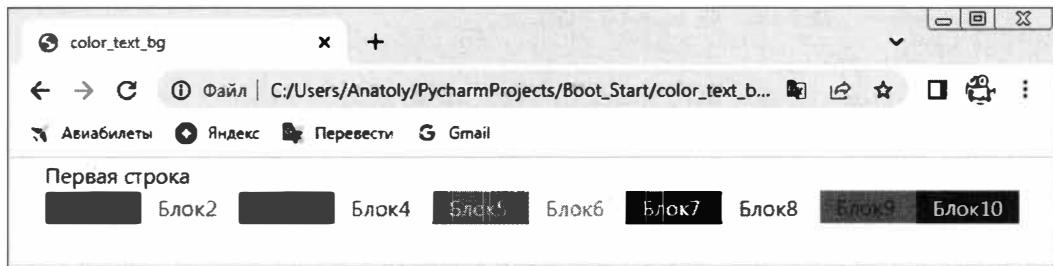


Рис. 3.14. Использование классов `text` и `bg` для задания цвету тексту и фону

### 3.7. Задание отступов элементам макета HTML-страниц

По умолчанию все адаптивные блоки «прилипают» друг к другу, однако достаточно часто требуется отделить их друг от друга дополнительным пространством. Особенно это актуально в тех случаях, когда они имеют рамки обрамления различных типов. Для задания отступов (разрывов) в Bootstrap существуют два элемента: `margin` и `padding`. Здесь `margin` (имеет сокращение — `m`) и `padding` (имеет сокращение — `p`) — это уникальные свойства CSS, которые добавляют отступы между элементами HTML-страницы и их содержимым. Свойство `margin` добавляет отступы за пределами элемента, а `padding` — внутри элемента (например, контейнера). Синтаксис для задания отступов:

`{property}{sides}-{size}`

Здесь приняты следующие обозначения:

- `property` — свойство;
- `sides` — сторона;
- `size` — размер.

Для свойства (`property`) в коде программ допустимы следующие сокращения:

- `t` — для классов `margin`, которые устанавливают отступы за пределами элемента;
- `p` — для классов `padding`, которые устанавливают отступы внутри элемента.

Параметр `сторона` (`sides`) может принимать сокращенные значения, которые устанавливают следующие отступы:

- `t` — отступ в верхней части элемента (`margin-top` ИЛИ `padding-top`);
- `b` — в нижней части элемента (`margin-bottom` ИЛИ `padding-bottom`);

- s — в стартовой части элемента (start), т. е. слева (margin-left или padding-left);
- e — в конечной части элемента (end), т. е. справа (margin-right или padding-right);
- x — с двух сторон элемента по горизонтали (\*-left и \*-right);
- y — с двух сторон элемента по вертикали (\*-top и \*-bottom);
- blank — отступ со всех четырех сторон элемента.

Параметр размер (size) может принимать следующие значения:

- 0 — нет отступов;
- 1 — отступ, равный \$spacer \* .25;
- 2 — отступ, равный \$spacer \* .5;
- 3 — отступ, равный \$spacer;
- 4 — отступ, равный \$spacer \* 1.5;
- 5 — отступ, равный \$spacer \* 3;
- auto — отступ, равный авто.

#### ПРИМЕЧАНИЕ

Здесь \$spacer = 1rem Rem — это единица типографики, равная корневому (базовому) значению размера шрифта (font-size). Это значит, что 1rem всегда будет равен значению font-size, которое было определено в HTML. Для большинства браузеров размер шрифта по умолчанию 16 px до тех пор, пока его не поменяют в настройках браузера (редко кто подобным занимается). Поэтому, как правило, \$spacer = 1rem=16px.

Пример HTML-кода страницы, в которой показано использование отступов для разделения адаптивных блоков, приведен в листинге 3.5, страница space\_1.html.

#### Листинг 3.5. Страница space\_1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>space_1</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container">
    <div class="row bg-success text-white">Первая строка</div>
    <div class="row">
        <div class="col mx-2 bg-primary text-white">Блок 1</div>
        <div class="col mx-2 bg-primary text-white">Блок 2</div>
        <div class="col mx-2 bg-primary text-white">Блок 3</div>
    </div>
</div>
</body>
</html>
```

Здесь для адаптивных блоков были вставлены горизонтальные отступы в две позиции (`mx-2`). В окне браузера данная страница будет иметь вид, представленный на рис. 3.15.

Как видно из данного рисунка, между адаптивными блоками появились горизонтальные отступы, однако между рядами (строками) нет разрывов, и они вплотную примыкают друг к другу. Исправим это и вставим отступы между рядами (листинг 3.6, страница `space_2.html`).

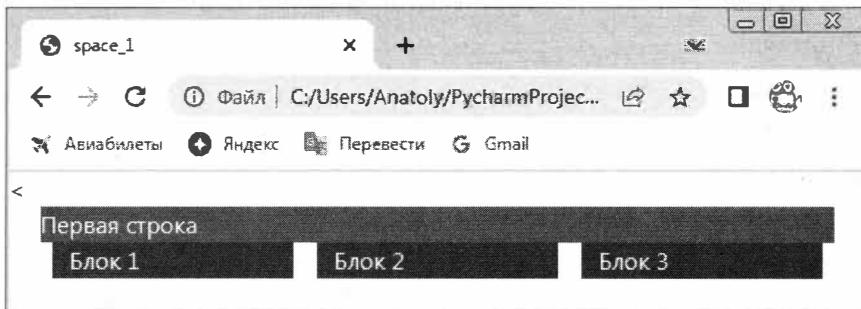


Рис. 3.15. Использование горизонтальных отступов между адаптивными блоками

#### Листинг 3.6. Страница `space_2.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>space_2</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container">
    <div class="row bg-success text-white">Первая строка</div>
    <div class="row my-2">
        <div class="col mx-2 bg-primary text-white">Блок 1</div>
        <div class="col mx-2 bg-primary text-white">Блок 2</div>
        <div class="col mx-2 bg-primary text-white">Блок 3</div>
    </div>
</div>
</body>
</html>
```

Здесь для второго ряда был вставлен вертикальный отступ в две позиции (`my-2`). В окне браузера эта страница будет иметь вид, представленный на рис. 3.16.

Как видно из данного рисунка, между рядами появился вертикальный отступ.

Теперь посмотрим, как можно создать отступы внутри элемента. Пример HTML-кода страницы, в которой показано использование отступов в теле адаптивных блоков, приведен в листинге 3.7, страница `space_3.html`.

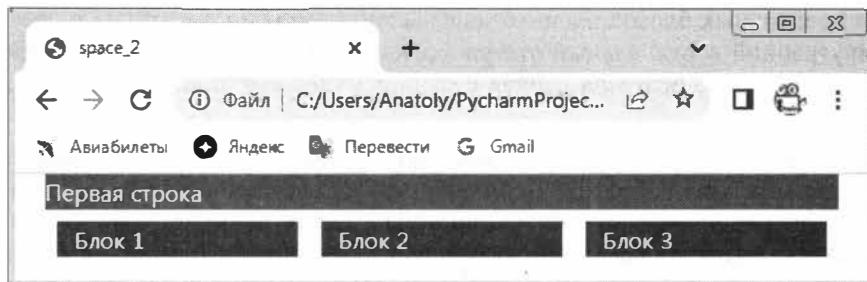


Рис. 3.16. Использование вертикальных отступов между рядами

**Листинг 3.7. Страница space\_3.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>space_3</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container">
    <div class="row bg-success text-white">Первая строка</div>
    <div class="row my-2">
        <div class="col mx-2 py-5 bg-primary text-white">Блок 1</div>
        <div class="col mx-2 py-5 bg-primary text-white">Блок 2</div>
        <div class="col mx-2 py-5 bg-primary text-white">Блок 3</div>
    </div>
</div>
</body>
</html>
```

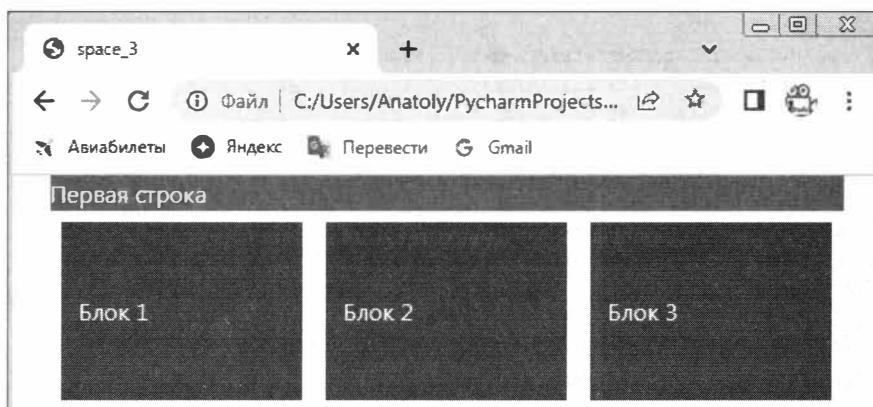


Рис. 3.17. Использование внутренних отступов в теле адаптивных блоков

Здесь для адаптивных блоков были вставлены горизонтальные отступы в две позиции (`mx-2`) и внутренний вертикальный отступ (между текстом и внешней границей) в пять позиций (`py-5`). В окне браузера данная страница будет иметь вид, представленный на рис. 3.17.

Как видно из рис. 3.17, между текстом внутри адаптивных блоков и их внешней границей появились отступы. Обратите внимание, что на всех этих HTML-страницах текст внутри адаптивных блоков по умолчанию прижимается к левому краю, т. е. к началу блока. Разберемся теперь, как можно выравнивать содержимое блоков HTML-страниц.

## 3.8. Выравнивание содержимого в адаптивных блоках HTML-страниц

Начнем с самого простого — выравнивания текста по горизонтальной оси (`x`). Для этого нужно использовать класс `text`, который обеспечивает следующие варианты выравнивания:

- `start` (умолчание браузера) — по левой стороне;
- `end` — по правой стороне;
- `center` — по центру.

Вот пример горизонтального выравнивания текста в программном коде:

```
class="text-start"  
class="text-center"  
class="text-end"
```

Пример HTML-кода страницы, в которой показано выравнивание текста в теле адаптивных блоков, приведен в листинге 3.8, страница `alignment_1.html`.

### Листинг 3.8. Страница `alignment_1.html`

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <title>alignment_1</title>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >  
    <script defer src="static/js/bootstrap.bundle.min.js"></script>  
</head>  
<body>  
    <div class="container text-center">  
        <div class="row bg-success text-white">  
            <div class="col">Блок в первой строке</div>  
        </div>  
        <div class="row my-2 text-white">  
            <div class="col mx-2 bg-primary">Блок 1</div>  
            <div class="col mx-2 bg-primary">Блок 2</div>  
            <div class="col mx-2 bg-primary">Блок 3</div>  
        </div>  
    </div>
```

```
</div>
</body>
</html>
```

Здесь выравнивание текста задано в родительском классе `container (text-center)`. В этом случае во всех дочерних элементах текст будет выровнен по центру. В окне браузера данная страница будет иметь вид, представленный на рис. 3.18.

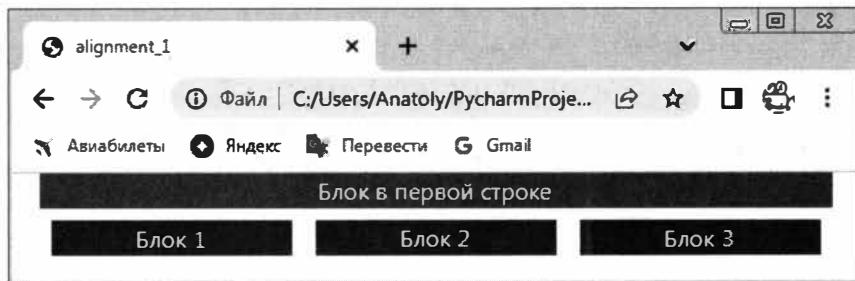


Рис. 3.18. Центрирование текста во всех аддаптивных блоках

Как видно из рис. 3.18, во всех дочерних элементах контейнера текст был выровнен по центру аддаптивного блока.

Немного изменим код этого примера и во втором ряду поменяем параметры выравнивания текста (листинг 3.9, страница `alignment_2.html`).

#### Листинг 3.9. Страница `alignment_2.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>alignment_2</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container text-center">
    <div class="row bg-success text-white">
        <div class="col">Блок в первой строке</div>
    </div>
    <div class="row my-2 text-white">
        <div class="col mx-2 bg-primary text-start">Блок 1</div>
        <div class="col mx-2 bg-primary">Блок 2</div>
        <div class="col mx-2 bg-primary text-end">Блок 3</div>
    </div>
</div>
</body>
</html>
```

Здесь выравнивание текста по центру задано в родительском классе `container (text-center)`. Однако во второй строке для первого блока задано выравнивание по левому краю (`text-start`), а для третьего блока — по правому краю (`text-end`). В окне браузера эта страница будет иметь вид, представленный на рис. 3.19.

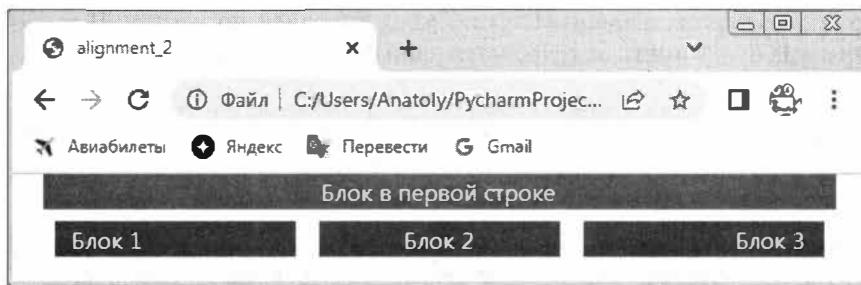


Рис. 3.19. Варианты выравнивания текста в адаптивных блоках

Как видно из данного рисунка, в адаптивных блоках текст выровнен по-разному: по центру, по левому и по правому краю.

### 3.9. Обозначение границ элементов макета HTML-страниц

В Bootstrap есть утилиты для формирования границ элементов макета, которые позволяют создать рамку, радиус закругления рамки, толщину и цвет рамки. Рамки отлично подходят для явного выделения блоков, для изображений, кнопок или любых других элементов интерфейса. Рамка может обрамлять элемент полностью или только одну из его сторон (рис. 3.20).

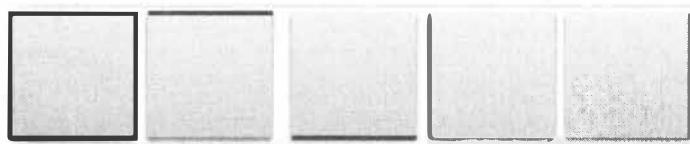


Рис. 3.20. Различные варианты обозначения границы элемента

Для добавления границ применяются следующие классы:

- `class="border"` — граница с четырех сторон;
- `class="border-top"` — граница сверху;
- `class="border-end"` — граница справа;
- `class="border-bottom"` — граница снизу;
- `class="border-start"` — граница слева.

Ширину рамки можно задать с помощью следующих классов:

- `class="border border-1";`
- `class="border border-2";`
- `class="border border-3";`

- class="border border-4";
- class="border border-5".

Пример HTML-кода страницы, в которой показано создание границ вокруг адаптивных блоков, приведен в листинге 3.10, страница border\_1.html.

#### Листинг 3.10. Страница border\_1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>border_1</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container text-center bg-primary">
    <div class="row bg-success text-white">
        <div class="col">Обозначение границ вокруг адаптивных блоков</div>
    </div>
    <div class="row my-2">
        <div class="col mx-2 bg-info border border-5">Блок 1</div>
        <div class="col mx-2 bg-info border-top border-5">Блок 2</div>
        <div class="col mx-2 bg-info border-bottom border-5">Блок 3</div>
        <div class="col mx-2 bg-info border-start border-5">Блок 4</div>
        <div class="col mx-2 bg-info border-end border-5">Блок 5</div>
    </div>
    <div class="row"></div>
</div>
</body>
</html>
```

Здесь созданы границы вокруг адаптивных блоков с разных сторон; цвет линии задан по умолчанию, а толщина — border-5. В окне браузера данная страница будет иметь вид, представленный на рис. 3.21.

Как видно из рис. 3.21, адаптивные блоки имеют ограничительные линии, расположенные с разных сторон.

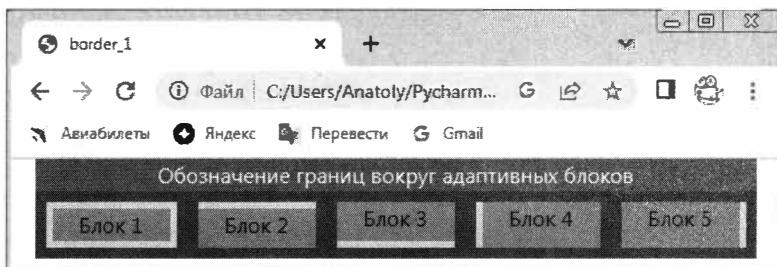


Рис. 3.21. Различные варианты обозначения границы вокруг адаптивного блока

Есть еще один способ обрамления адаптивных блоков, когда можно убирать границу с одной из сторон. Для этих целей используется следующий класс:

- class="border-0" — убирается ограничительная линия с четырех сторон;
- class="border-top-0" — убирается ограничительная линия сверху;
- class="border-end-0" — убирается ограничительная линия справа;
- class="border-bottom-0" — убирается ограничительная линия снизу;
- class="border-start-0" — убирается ограничительная линия слева.

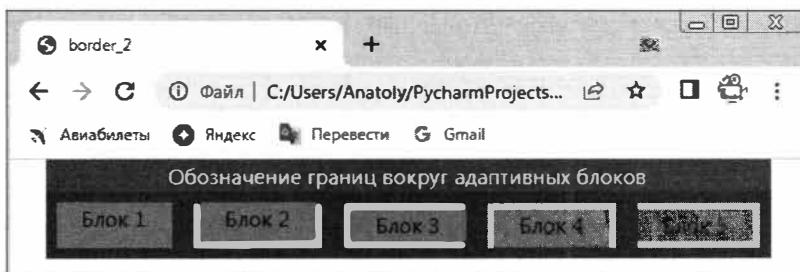
Пример HTML-кода страницы, в которой показано создание границ вокруг адаптивных блоков путем удаления ограничительной линии с одной из сторон, приведен в листинге 3.11, страница border\_2.html.

#### Листинг 3.11. Страница border\_2.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>border_2</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <div class="container text-center bg-primary">
        <div class="row bg-success text-white">
            <div class="col">Обозначение границ вокруг адаптивных блоков</div>
        </div>
        <div class="row my-2">
            <div class="col mx-2 bg-info border border-0">Блок 1</div>
            <div class="col mx-2 bg-info border border-5 border-top-0">Блок 2</div>
            <div class="col mx-2 bg-info border border-5 border-end-0">Блок 3</div>
            <div class="col mx-2 bg-info border border-5 border-bottom-0">
                Блок 4</div>
            <div class="col mx-2 bg-info border border-5 border-start-0">
                Блок 5</div>
        </div>
        <div class="row"></div>
    </div>
</body>
</html>
```

Здесь заданы границы вокруг адаптивных блоков с разных сторон. Цвет линии установлен по умолчанию, а толщина — border-5. В окне браузера эта страница будет иметь вид, представленный на рис. 3.22.

Как видно из данного рисунка, ограничительную линию можно удалить либо со всех сторон адаптивного блока, либо с одной из сторон.



**Рис. 3.22.** Различные варианты обозначения границы вокруг адаптивного блока путем удаления ограничительной линии с одной из сторон

Изменить цвет границы вокруг адаптивного блока можно с помощью следующих классов:

- |   |   |
|---|---|
| <input type="checkbox"/> class="border border-primary";   | <input type="checkbox"/> class="border border-info";  |
| <input type="checkbox"/> class="border border-secondary"; | <input type="checkbox"/> class="border border-light"; |
| <input type="checkbox"/> class="border border-success";   | <input type="checkbox"/> class="border border-dark";  |
| <input type="checkbox"/> class="border border-danger";    | <input type="checkbox"/> class="border border-muted"; |
| <input type="checkbox"/> class="border border-warning";   | <input type="checkbox"/> class="border border-white". |

Пример HTML-кода страницы, в которой показано создание границ вокруг адаптивных блоков разного цвета, приведен в листинге 3.12, страница border\_color.html.

#### Листинг 3.12. Страница border\_color.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>border_1</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container text-center">
    <div class="row bg-success text-white">
        <div class="col">Обозначение границ вокруг адаптивных блоков</div>
    </div>
    <div class="row my-2">
        <div class="col mx-2 bg-info border border-5 border-primary">
            Блок 1</div>
        <div class="col mx-2 bg-info border border-5 border-secondary">
            Блок 2</div>
        <div class="col mx-2 bg-info border border-5 border-success">
            Блок 3</div>
        <div class="col mx-2 bg-info border border-5 border-danger">
            Блок 4</div>
    </div>
</div>
```

```

<div class="col mx-2 bg-info border border-5 border-warning">
    Блок 5</div>
<div class="col mx-2 bg-primary border border-5 border-info">
    Блок 6</div>
<div class="col mx-2 bg-info border border-5 border-light">Блок 7</div>
<div class="col mx-2 bg-info border border-5 border-dark">Блок 8</div>
<div class="col mx-2 bg-info border border-5 border-muted">Блок 9</div>
<div class="col mx-2 bg-info border border-5 border-white">Блок 10</div>
</div>
<div class="row"></div>
</div>
</body>
</html>

```

Здесь заданы цвета рамок границы вокруг адаптивных блоков с разных сторон толщиной линии border-5. В окне браузера данная страница будет иметь вид, представленный на рис. 3.23.

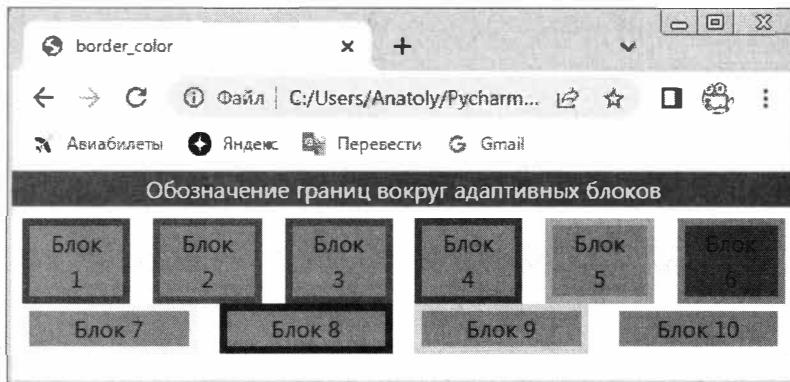


Рис. 3.23. Различные варианты цвета границы вокруг адаптивных блоков

Как видно из рис. 3.23, для ограничительных рамок можно задавать разные цвета. Обратите внимание на блоки 7 и 10, на которых не видно ограничительных рамок. Это связано с тем, что ограничительные рамки имеют тот же цвет, что и фон.

Углы ограничительных рамок можно округлить, для этого предусмотрены следующие классы:

- class="rounded";
- class="rounded-top";
- class="rounded-end";
- class="rounded-bottom";
- class="rounded-start";
- class="rounded-circle";
- class="rounded-pill".

Пример HTML-кода страницы, в которой показано создание рамок с округленными углами, приведен в листинге 3.13, страница border\_radius.html.

**Листинг 3.13. Страница border\_radius.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>border_radius</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container text-center">
    <div class="row bg-success text-white">
        <div class="col">Обозначение границ вокруг адаптивных блоков</div>
    </div>
    <div class="row my-2">
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded">Блок 1</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-top">Блок 2</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-end">Блок 3</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-bottom">Блок 4</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-start">Блок 5</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-circle">Блок 6</div>
        <div class="col mx-2 bg-warning border border-3
            border-primary rounded-pill">Блок 7</div>
    </div>
    <div class="row"></div>
</div>
</body>
</html>
```

Здесь заданы рамки вокруг адаптивных блоков с толщиной линии border-3, при этом реализованы разные варианты округленных углов. В окне браузера данная страница будет иметь вид, представленный на рис. 3.24.

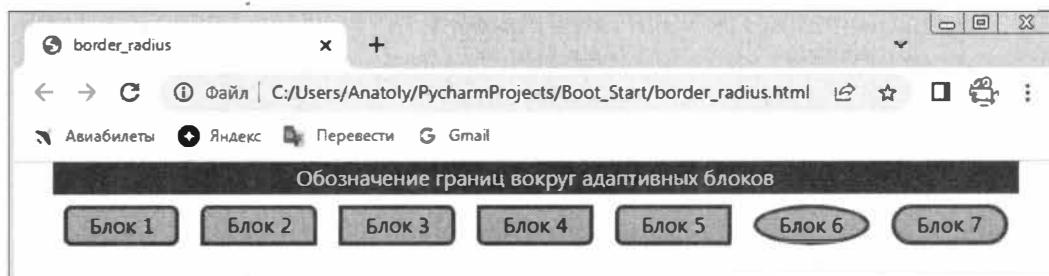


Рис. 3.24. Различные варианты округленных углов рамок вокруг адаптивного блока

Можно задать радиус округления углов ограничительных рамок, для этого используются следующие классы:

- class="rounded-0";
- class="rounded-1";
- class="rounded-2";
- class="rounded-3";
- class="rounded-4";
- class="rounded-5".

Пример HTML-кода страницы, в которой показано создание рамок с разными радиусами округления углов, приведен в листинге 3.14, страница border\_radius\_1.html.

#### Листинг 3.14. Страница border\_radius\_1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>border_radius_1</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container text-center">
    <div class="row bg-success text-white">
        <div class="col">Обозначение границ вокруг адаптивных блоков</div>
    </div>
    <div class="row my-2">
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-0">Блок 1</div>
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-1">Блок 2</div>
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-2">Блок 3</div>
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-3">Блок 4</div>
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-4">Блок 5</div>
        <div class="col mx-2 bg-warning border border-3 border-primary rounded-5">Блок 6</div>
    </div>
    <div class="row"></div>
</div>
</body>
</html>
```

Здесь созданы рамки вокруг адаптивных блоков с толщиной линии border-3, при этом заданы разные радиусы округленных углов. В окне браузера данная страница будет иметь вид, представленный на рис. 3.25.

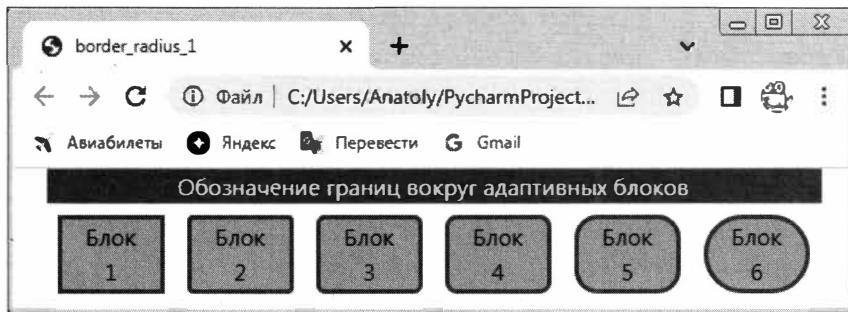


Рис. 3.25. Различные варианты радиусов округления углов рамок вокруг адаптивного блока

## 3.10. Пример использования адаптивных контейнеров

На HTML-странице может быть несколько адаптивных контейнеров. Для создания адаптивного контейнера с фиксированной шириной предусмотрен класс `container`. В качестве примера создадим HTML-страницу с двумя адаптивными контейнерами с фиксированной шириной (листинг 3.15, страница `start.html`).

### Листинг 3.15. Страница `start.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Start</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container p-3 bg-primary text-white text-center">
    <h1>Это страница Bootstrap</h1>
    <p>Измените размер этой адаптивной страницы, чтобы увидеть эффект!</p>
</div>
<div class="container mt-3 text-center border border-3">
    <div class="row">
        <div class="col-sm-4 ">
            <h3>Колонка 1</h3>
            <p>Это содержимое</p>
            <p>первой колонки</p>
        </div>
        <div class="col-sm-4 ">
            <h3>Колонка 2</h3>
            <p>Это содержимое</p>
            <p>второй колонки</p>
        </div>
    </div>
</div>
```

```
<div class="col-sm-4 ">
    <h3>Колонка 3</h3>
    <p>Это содержимое</p>
    <p>третьей колонки</p>
</div>
</div>
</body>
</html>
```

На этой странице имеются два контейнера класса `container`. В первом контейнере находится текст, во втором — создан один ряд, в котором имеются три аддаптивных блока с вложенным в них текстом. В окне браузера данная страница будет иметь вид, представленный на рис. 3.26.

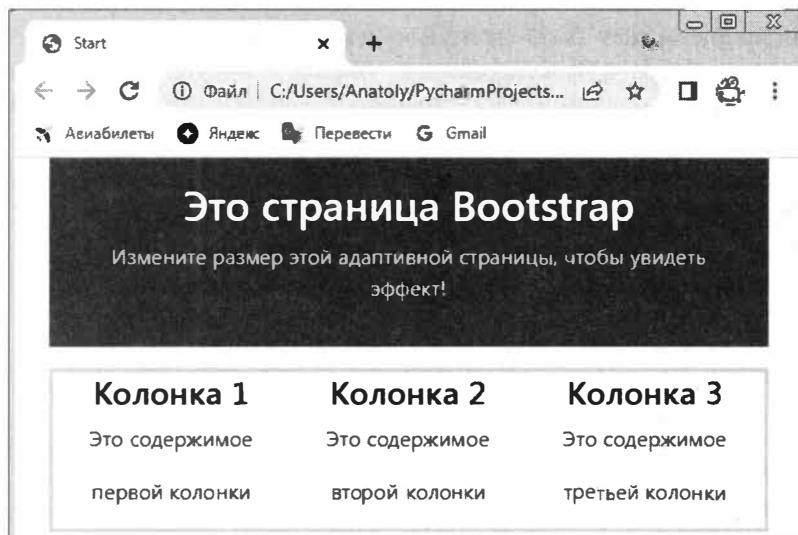


Рис. 3.26. Страница с двумя контейнерами с фиксированной шириной класса `container`

Обратите внимание, что при изменении ширины окна браузера меняется ширина контейнера и их содержимого, при этом окно браузера всегда шире, чем ширина контейнеров (слева и справа остается свободное пространство). При минимальной ширине окна браузера колонки трансформируются в три строки, и боковые отступы исчезают, при этом размер шрифта не уменьшается (рис. 3.27).

В таком виде страница будет отчетливо отображаться как на экране компьютера, так и на экране смартфона, текст будет оставаться разборчивым и читаемым.

Теперь создадим HTML-страницу с двумя аддаптивными подвижными, или «резиновыми» контейнерами на основе класса `container-fluid` (листинг 3.16, страница `start_1.html`).

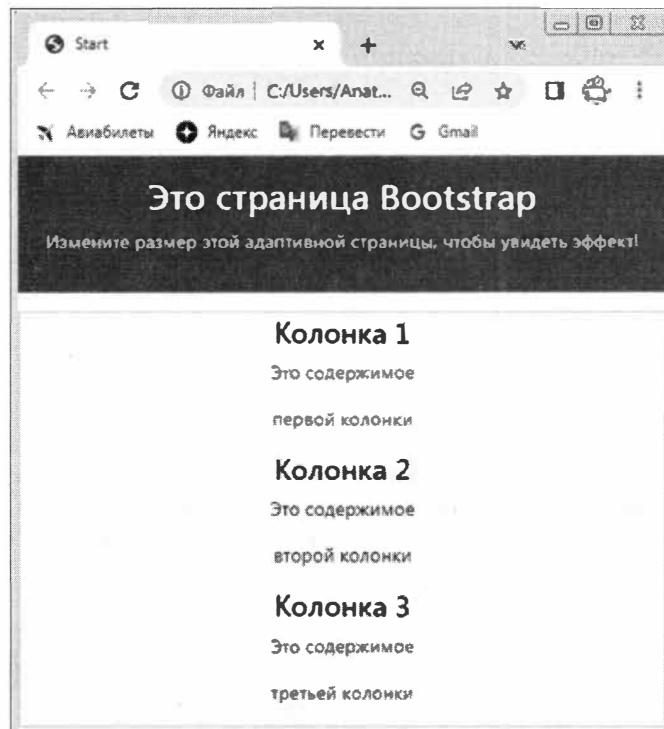


Рис. 3.27. Страница с двумя контейнерами с фиксированной шириной класса container при минимальном размере окна браузера

#### Листинг 3.16. Страница start\_1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Start</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<div class="container-fluid p-3 bg-primary text-white text-center">
    <h1>Это страница Bootstrap</h1>
    <p>Измените размер этой адаптивной страницы, чтобы увидеть эффект!</p>
</div>

<div class="container-fluid mt-3 text-center border border-3">
    <div class="row">
        <div class="col-sm-4 ">
            <h3>Колонка 1</h3>
            <p>Это содержимое</p>
            <p>первой колонки</p>
        </div>
```

```
<div class="col-sm-4 ">
    <h3>Колонка 2</h3>
    <p>Это содержимое</p>
    <p>второй колонки</p>
</div>
<div class="col-sm-4 ">
    <h3>Колонка 3</h3>
    <p>Это содержимое</p>
    <p>третьей колонки</p>
</div>
</div>
</body>
</html>
```

На этой странице имеются два контейнера класса `container-fluid`. В первом контейнере находится текст, во втором — создан один ряд, в котором имеются три адаптивных блока с вложенным в них текстом. В окне браузера данная страница будет иметь вид, представленный на рис. 3.28.

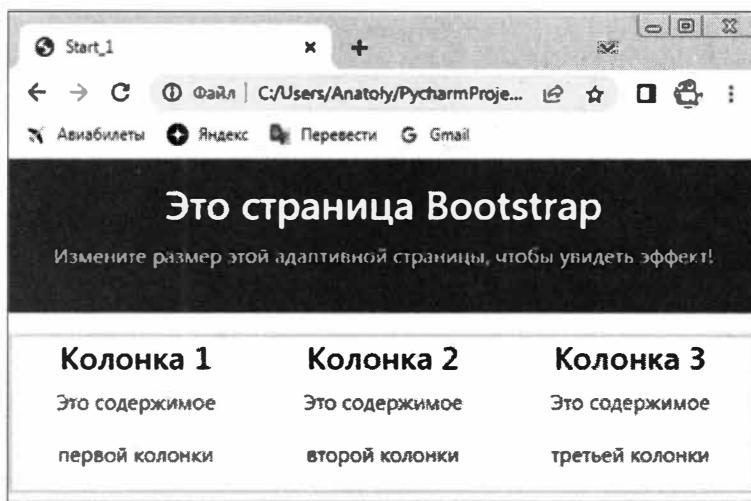


Рис. 3.28. Страница с двумя адаптивными подвижными контейнерами класса `container-fluid`

Обратите внимание, что при изменении ширины браузера меняется ширина контейнера и их содержимого, при этом ширина контейнеров всегда соответствует ширине окна браузера (слева и справа нет свободного пространства). При минимальной ширине браузера колонки трансформируются в три строки.

На этом мы закончим знакомство с макетированием HTML-страниц на основе фреймворка Bootstrap. Конечно, в Bootstrap имеется множество различных визуальных элементов для создания привлекательного веб-интерфейса, с которыми вы можете познакомиться в специальной литературе. Некоторые из этих элементов мы будем применять в шаблонах HTML-страниц и более детально познакомимся с ними в следующих главах на конкретных примерах.

## 3.11. Таблицы Bootstrap

Таблицы Bootstrap можно использовать как для вывода данных, так и для макетирования страниц. В базовом варианте таблицы имеют небольшие отступы и горизонтальные разделители. Для того чтобы создать таблицу, понадобятся следующие теги:

```
<table class="table">
  <thead>
    <tr>
      <th>...</th>
      ...
      <th>...</th>
    </tr>
  </thead>

  <tbody>
    <tr>
      <td>...</td>
      ...
      <td>...</td>
    </tr>
  </tbody>

</table>
```

← Таблица  
← Заголовок таблицы  
← Стока заголовка  
← Колонка заголовка  
← Колонка заголовка  
← Колонка заголовка  
← Тело таблицы  
← Стока в теле таблицы  
← Колонка в теле таблицы  
← Колонка в теле таблицы  
← Колонка в теле таблицы

Таблицу создают с помощью следующих тегов:

- <table> — создает таблицу;
- <thead> — заголовок таблицы;
- <tbody> — тело таблицы;
- <tr> — строку;
- <td> — колонку.

В качестве примера создадим HTML-страницу с таблицей, состоящей из трех строк и трех колонок (листинг 3.17, страница table.html).

### Листинг 3.17. Страница table.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Таблица</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="static/css/bootstrap.min.css" >
  <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<table class="table">
  <thead>
```

```

<tr>
    <th>№</th>
    <th>Фамилия</th>
    <th>Имя</th>
    <th>E-mail</th>
</tr>
</thead>
<tbody>
    <tr>
        <td>1</td>
        <td>Иванов</td>
        <td>Иван</td>
        <td>ivan@mail.ru</td>
    </tr>
    <tr>
        <td>2</td>
        <td>Петров</td>
        <td>Петр</td>
        <td>petr@mail.ru</td>
    </tr>
    <tr>
        <td>3</td>
        <td>Сидоров</td>
        <td>Семен</td>
        <td>sid@mail.ru</td>
    </tr>
</tbody>
</table>
</body>
</html>

```

Здесь нет контейнеров, и таблица находится непосредственно в теле HTML-страницы. В окне браузера данная страница будет иметь вид, представленный на рис. 3.29.

Как видно из данного рисунка, с параметрами по умолчанию таблица представлена в черно-белом цвете, имеет выделенный заголовок и горизонтальные линии, разделяющие строки.

The screenshot shows a web browser window with a title bar 'Таблица'. The address bar displays the path 'C:/Users/Anatoly/P...'. Below the address bar are several icons: 'Авиабилеты', 'Яндекс', 'Перевести', and 'G Gmail'. The main content area contains a table with the following data:

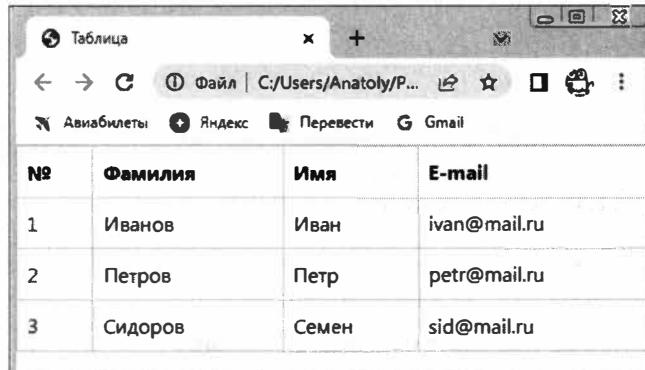
№	Фамилия	Имя	E-mail
1	Иванов	Иван	ivan@mail.ru
2	Петров	Петр	petr@mail.ru
3	Сидоров	Семен	sid@mail.ru

Рис. 3.29. Таблица Bootstrap

Для того чтобы создать для таблицы границы, используется класс `table-bordered`. Изменим в листинге 3.17 следующим образом строку с классом `table`:

```
<table class="table table-bordered">
```

После этого таблица примет вид, представленный на рис. 3.30.



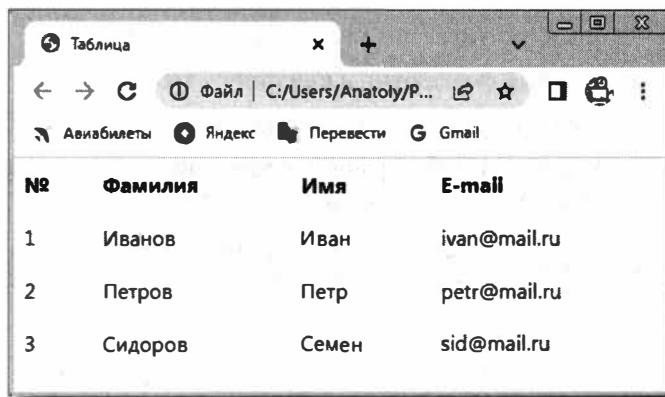
№	Фамилия	Имя	E-mail
1	Иванов	Иван	ivan@mail.ru
2	Петров	Петр	petr@mail.ru
3	Сидоров	Семен	sid@mail.ru

Рис. 3.30. Таблица Bootstrap с границами ячеек

Если таблица используется для макетирования страницы, то все границы можно удалить, для чего предусмотрен класс `table-borderless`. Для удаления границ изменим следующим образом строку с классом `table` в листинге 3.17:

```
<table class="table table-borderless">
```

После этого таблица примет вид, представленный на рис. 3.31.



№	Фамилия	Имя	E-mail
1	Иванов	Иван	ivan@mail.ru
2	Петров	Петр	petr@mail.ru
3	Сидоров	Семен	sid@mail.ru

Рис. 3.31. Таблица Bootstrap без границ ячеек

Считается хорошим тоном, когда четные и нечетные строки таблицы выделены альтернативными цветами. В Bootstrap это можно сделать с помощью класса `table-striped`. Изменим в листинге 3.17 следующим образом строку с классом `table`:

```
<table class="table table-striped">
```

После этого таблица примет вид, представленный на рис. 3.32.

№	Фамилия	Имя	E-mail
1	Иванов	Иван	ivan@mail.ru
2	Петров	Петр	petr@mail.ru
3	Сидоров	Семен	sid@mail.ru

Рис. 3.32. Таблица Bootstrap со строками альтернативного цвета

В таблицах можно использовать контекстные классы для окрашивания всей таблицы (`<table>`), строк таблицы (`<tr>`) или ячеек таблицы (`<td>`):

- `table-primary` — синий, указывает на важное действие;
- `table-success` — зеленый, указывает на успешное или положительное действие;
- `table-danger` — красный, указывает на опасное или потенциально негативное действие;
- `table-info` — голубой, указывает на нейтральное информативное изменение или действие;
- `table-warning` — оранжевый, указывает на предупреждение;
- `table-active` — серый, применяется при наведении курсора мыши на строку или ячейку таблицы;
- `table-secondary` — серый, указывает на менее важное действие;
- `table-light` — светло-серый фон таблицы или строки таблицы;
- `table-dark` — черный фон таблицы или строки таблицы.

В качестве примера создадим HTML-страницу с таблицей, в которой показаны классы с основными цветами (листинг 3.18, страница `table_1.html`).

#### Листинг 3.18. Страница `table_1.html`

```

<!DOCTYPE html>
<html lang="en">
<head>
    <title>Таблица</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="static/css/bootstrap.min.css" >
    <script defer src="static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
<table class="table">
    <thead>

```

```
<tr>
    <th>Цвет</th>
    <th>Класс</th>
</tr>
</thead>
<tbody>
    <tr>
        <td>Default</td>
        <td>Default</td>
    </tr>
    <tr class="table-primary">
        <td>Primary</td>
        <td>class="table-primary"</td>
    </tr>
    <tr class="table-success">
        <td>Success</td>
        <td>class="table-success"</td>
    </tr>
    <tr class="table-danger">
        <td>Danger</td>
        <td>class="table-danger"</td>
    </tr>
    <tr class="table-info">
        <td>Info</td>
        <td>class="table-info"</td>
    </tr>
    <tr class="table-warning">
        <td>Warning</td>
        <td>class="table-warning"</td>
    </tr>
    <tr class="table-active">
        <td>Active</td>
        <td>class="table-active"</td>
    </tr>
    <tr class="table-secondary">
        <td>Secondary</td>
        <td>class="table-secondary"</td>
    </tr>
    <tr class="table-light">
        <td>Light</td>
        <td>class="table-light"</td>
    </tr>
    <tr class="table-dark">
        <td>Dark</td>
        <td>class="table-dark"</td>
    </tr>
</tbody>
</table>
</body>
</html>
```

В окне браузера данная страница будет иметь вид, представленный на рис. 3.33.

The screenshot shows a web browser window titled 'Таблица'. The address bar displays 'Файл | C:/Users/Anat...'. Below the address bar, there are links for 'Авиабилеты', 'Яндекс', 'Перевести', and 'Gmail'. The main content is a table with two columns: 'Цвет' (Color) and 'Класс' (Class). The rows are colored in a gradient: Default (light gray), Primary (medium gray), Success (dark gray), Danger (very dark gray), Info (light gray), Warning (medium gray), Active (dark gray), Secondary (light gray), Light (light gray), and Dark (black). Each row also contains its corresponding Bootstrap class value.

Цвет	Класс
Default	Default
Primary	class="table-primary"
Success	class="table-success"
Danger	class="table-danger"
Info	class="table-info"
Warning	class="table-warning"
Active	class="table-active"
Secondary	class="table-secondary"
Light	class="table-light"
Dark	class="table-dark"

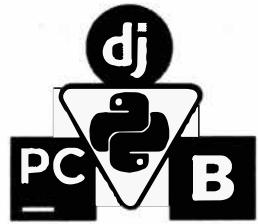
Рис. 3.33. Таблица Bootstrap со строками разного цвета

## 3.12. Краткие итоги

Итак, мы познакомились с некоторыми базовыми элементами фреймворка Bootstrap, которые нам пригодятся при формировании шаблонов HTML-страниц. Но прежде чем приступить к разработке веб-приложений с использованием Python, нужно познакомиться с фреймворком Django. В частности, нам предстоит узнать:

- историю появления фреймворка Django и его развитие;
- структуру веб-приложений на Django;
- механизм доступа к базам данных;
- механизм формирования динамических HTML-страниц на основе шаблонов.

И, наконец, получить практические навыки создания элементарных веб-приложений с помощью Django. Рассмотрению этих вопросов и посвящена следующая глава.



## ГЛАВА 4

# Знакомимся с фреймворком Django

В этой главе мы ответим на вопрос «Что такое Django?» и поясним, почему этот фреймворк столь привлекателен для программистов. Мы также покажем вам некоторые основные базовые блоки приложения, написанного с помощью Django (хотя у вас пока еще не будет среды разработки и тестирования). В общих чертах мы познакомимся со структурой и основными элементами Django, структурой приложений на нем, основными понятиями и определениями. В процессе изучения материала главы:

- будет создан первый проект на Django;
- рассмотрены понятия о «представлениях» и «маршрутизации» при разработке веб-приложений на Django.

Итак, приступим к знакомству с основами работы на Django.

### 4.1. Общие представления о Django

Django — это фреймворк, написанный на Python и для Python, позволяющий быстро создавать безопасные и удобно поддерживаемые веб-сайты. Разработанный опытными специалистами, Django берет на себя большую часть работы, поэтому разработчик может сосредоточиться на написании своего веб-приложения без необходимости изобретать велосипед. Django — это бесплатный фреймворк с открытым исходным кодом, он имеет активное сообщество, отличную документацию и множество вариантов как бесплатной, так и коммерческой поддержки.

Django представляет собой весьма эффективный инструментарий, который имеет следующие характерные особенности.

**Полнокомплектный инструмент.** Django следует философии «всё в одном флаконе» и предоставляет почти всё необходимое, что может понадобиться разработчикам для своих проектов. Поскольку все, что нужно, является частью единого продукта, все модули безупречно работают вместе в соответствии с последовательными принципами проектирования. Для него также имеется подробная документация.

**Универсальный инструмент.** С помощью Django можно создавать практически любые типы веб-сайтов — от систем управления контентом до социальных сетей и новостных порталов. Он может работать с любой клиентской средой и способен доставлять контент практически в любом формате (HTML, RSS-каналы, JSON, XML и т. д.).

На Django можно реализовать практически любую функциональность, которая может понадобиться конечному пользователю, он работает с различными популярными базами данных, при необходимости его базовые модули могут быть дополнены сторонними компонентами.

**Инструмент для разработки безопасных приложений.** Django помогает разработчикам избежать многих распространенных ошибок безопасности, предоставляя фреймворк с автоматической защитой сайта. Так, в Django реализован безопасный способ управления учетными записями пользователей и паролями, что предотвращает появление распространенных ошибок — таких, например, как размещение информации о сеансе в файлах cookie, где она уязвима. В Django файлы cookie содержат только ключ, а содержательная информация хранится отдельно в базе данных. Все пароли хранятся только в зашифрованном или хешированном виде. Как известно, хешированный пароль — это пароль фиксированной длины, созданный путем его обработки через криптографическую хеш-функцию. Django может проверить правильность введенного пароля, пропустив его через хеш-функцию и сравнив вывод с сохраненным значением хеша. Благодаря «одностороннему» характеру функции, даже если сохраненное хеш-значение скомпрометировано, злоумышленнику будет сложно определить исходный пароль.

**Масштабируемые приложения.** Django использует компонентную архитектуру, при которой каждая часть создаваемого приложения независима от других частей и, следовательно, может быть заменена либо изменена. Четкое разделение между частями означает, что Django может масштабироваться при увеличении трафика путем добавления оборудования на любом уровне: серверы кеширования, серверы баз данных или серверы приложений. Один из самых загруженных сайтов — Instagram — успешно масштабирован на Django.

**Разработанные приложения удобны в сопровождении.** Код Django написан на основе принципов и шаблонов проектирования, которые поощряют создание поддерживающего и повторно используемого кода. В частности, в нем задействован принцип DRY (Don't Repeat Yourself, не повторяйся), что позволяет избежать ненужного дублирования и сокращает объем кода. Django также способствует группировке связанных функциональных возможностей в повторно используемые приложения и группирует связанный программный код в модули в соответствии с концепцией MVC (Model–View–Controller).

#### ПРИМЕЧАНИЕ

Аббревиатуру MVC или Model–View–Controller (можно перевести как «Модель–Представление–Контроллер» или «Модель–Вид–Контроллер») — это схема разделения приложения на три отдельных компонента: модель (описывает структуру данных), представление (отвечает за отображение данных) и контроллер (интерпретирует действия пользователя). Таким образом, в разработанном приложении модификация каждого компонента может осуществляться независимо друг от друга.

**Разработанные приложения являются кроссплатформенными.** Django написан на Python, который работает на многих платформах. Это означает, что вы не привязаны к какой-либо конкретной серверной платформе и можете запускать приложения на многих версиях Linux, Windows и macOS. Кроме того, Django хорошо поддерживается многими веб-хостингами, которые часто предоставляют определенную инфраструктуру и документацию для размещения сайтов Django. Хотя следует отметить, что разверты-

вание готового сайта на хостинге внешнего провайдера — не совсем простая процедура, которая требует определенных знаний и квалификации.

Django был разработан в период с 2003 по 2005 год командой, которая занималась созданием и обслуживанием веб-сайтов для газет. Фреймворк был так назван в честь джазового музыканта цыганского происхождения Джанго Рейнхардта, который виртуозно импровизировал на многих музыкальных инструментах. Создав несколько сайтов, команда начала повторно использовать общий код и шаблоны. В итоге в июле 2005 года общий код эволюционировал в Open Source — проект веб-фреймворка Django.

Django продолжает расти и улучшаться с момента появления первого релиза (1.0), вышедшего в сентябре 2008 года, до недавно выпущенной версии 4.1.4. В каждой последующей версии были исправлены обнаруженные ошибки и добавлены новые функциональные возможности, начиная от поддержки новых типов баз данных, шаблонизаторов и кеширования до добавления «общих» функций и классов, уменьшающих объем кода, который разработчики должны писать для ряда задач.

Django — это процветающий совместный проект с открытым исходным кодом, в котором заняты многие тысячи пользователей и участников. Несмотря на то что у него все еще есть некоторые особенности, отражающие его происхождение, Django превратился в универсальную среду для разработки веб-сайтов любого типа.

Django является весьма популярным фреймворком, он использовался при создании таких крупных сайтов, как Instagram, Mozilla, The Washington Times, YouTube, Google и др. На рынке инструментальных средств, которые предназначены для разработки веб-приложений, лидируют два фреймворка: Django (на основе языка Python) и Laravel (на PHP). Поскольку Laravel основан на PHP, то он более сложен в изучении и освоении. Синтаксис PHP похож на C, C ++ и Java, он чувствителен к регистру для имен переменных и требует наличия точек с запятой для завершения операторов. Для указания метода необходимы фигурные скобки, специальные операторы и символы. Эти атрибуты PHP делают Laravel более сложным для изучения и освоения по сравнению с Django, основанном на Python.

Впрочем, нет никаких окончательных оценок популярности серверных фреймворков, хотя сайты, подобные Hot Framework, пытаются оценить их популярность, задействуя такие механизмы, как подсчет количества проектов на GitHub для каждой платформы. На сегодняшний день среди фреймворков Django занимает лидирующие позиции.

#### **ПРИМЕЧАНИЕ**

GitHub — это система управления проектами и версиями программного кода, а также общедоступная платформа, созданная для разработчиков. GitHub позволяет программистам разместить на нем свой программный код и дает возможность каждому разработчику обмениваться своими достижениями с другими людьми по всему миру.

## **4.2. Структура приложений на Django**

В типовом информационном сайте веб-приложение ожидает HTTP-запросы от веб-браузера пользователя. Получив запрос, приложение обрабатывает его и выполняет запограммированные действия. Затем приложение возвращает ответ веб-браузеру пользователя. Возвращаемая страница может быть либо статической, либо динамической. В последнем случае некоторые данные вставляются в HTML-шаблон.

Веб-приложение, написанное на Django, разбито на четыре базовых блока, которые содержатся в отдельных файлах, независимых друг от друга, но при этом работающих в связке.

Фреймворк Django реализует архитектуру Model–View–Template, или, сокращенно, MVT, которая по факту является модификацией распространенной в веб-программировании архитектуры MVC (Model–View–Controller). Схематично архитектура MVT в Django представлена на рис. 4.1.

Приложение на Django состоит из следующих блоков:

- диспетчер URL-адресов (URL-mapper или картостроитель адресов);
- представления (View);
- модели (Models);
- шаблоны (Templates).

Рассмотрим эти блоки подробнее.

**Диспетчер URL-адресов (URLs).** Теоретически можно обрабатывать запросы с каждого URL-адреса с помощью одной функции, но на практике гораздо удобнее писать отдельную функцию для обработки каждого ресурса. URL-mapper служит для перенаправления HTTP-запросов в соответствующее представление (View) на основе URL-адреса запроса. URL-mapper также может извлекать данные из URL-адреса в соответствии с заданным шаблоном и передавать их в соответствующую функцию в виде аргументов.

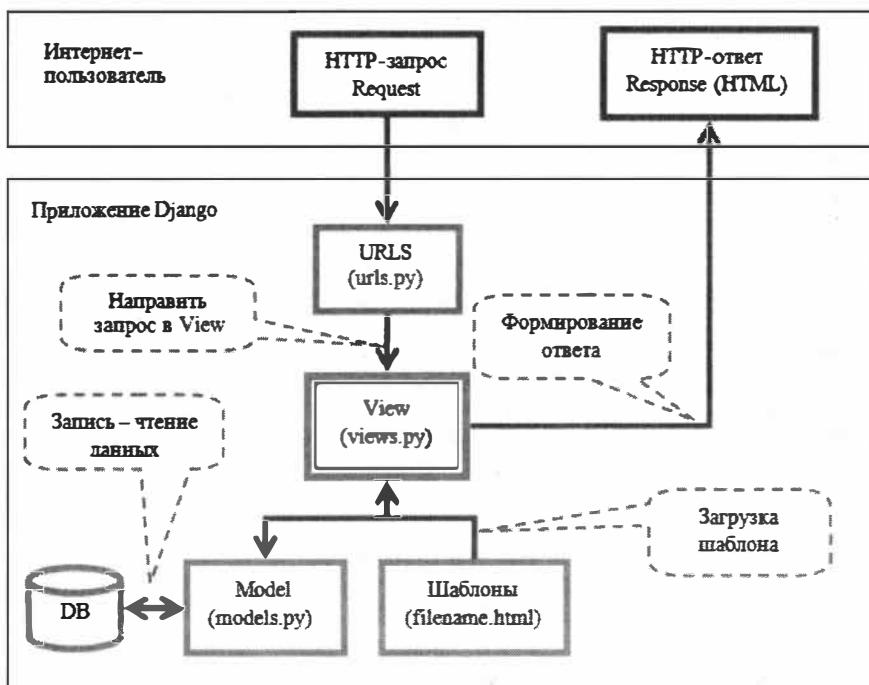


Рис. 4.1. Базовые блоки веб-приложения на Django

**Модели (Models).** Модели представляют собой объекты Python, которые определяют структуру данных приложения и предоставляют механизмы для управления данными (добавления, изменения, удаления) и выполнения запросов к базе данных.

**Шаблоны (Templates).** Template (шаблон) — это текстовый файл, определяющий структуру или разметку страницы (например, HTML-страницы), с полями, которые используются для подстановки актуального содержимого из базы данных или параметров, вводимых пользователем.

**Представление (View).** Центральная роль в этой архитектуре принадлежит представлению (view). Когда к приложению Django приходит запрос от удаленного пользователя (HTTP-запрос), то URL-диспетчер определяет, с каким ресурсом нужно сопоставить этот запрос, и передает его выбранному ресурсу. Ресурсом в этом случае является представление (view), которое, получив запрос, обрабатывает его определенным образом. В процессе обработки запроса представление (view) может обращаться через модели (model) к базе данных (БД), получать из нее данные или, наоборот, сохранять информацию в БД. Результат обработки запроса отправляется обратно, и этот результат пользователь видит в своем браузере. Как правило, результат обработки запроса представляет собой генерированный HTML-код, для генерации которого применяются шаблоны (template).

Лучший помощник при изучении нового материала — практика. Поэтому продолжим изучение Django путем реализации с нуля своего первого веб-приложения.

## 4.3. Первый проект на Django

Следует напомнить, что мы будем работать в среде PyCharm с проектом `Web_1`, который был создан в главе 1. Нужно иметь в виду, что при установке Django в папке виртуальной среды автоматически устанавливается скрипт `django-admin.py`, а в Windows — также исполняемый файл `django-admin.exe`. Их можно найти в папке виртуальной среды, в которую проводилась установка Django: на Windows — в подкаталоге `Scripts` (рис. 4.2), а на Linux/macOS — в каталоге `bin`.

Исполняемый файл `django-admin.exe` предоставляет возможность выполнения ряда команд для управления проектом Django. В частности, для создания проекта служит команда `startproject`, которой в качестве аргумента передается название проекта.

Итак, загрузите PyCharm и откройте проект `Web_1`, который был создан в главе 1. Теперь можно войти в окно терминала PyCharm и создать новый проект Django с именем `hello`. Для этого в окне терминала выполните следующую команду (рис. 4.3):

```
django-admin startproject hello
```

После выполнения этой команды в папке `Web_1` проекта PyCharm будет создана новая папка `hello` проекта Django (рис. 4.4).

Все веб-проекты на Django имеют одинаковую структуру. Изучим более детально эту структуру на примере нашего проекта `hello` (рис. 4.5), последовательно рассмотрев каждую папку и файл начального проекта, который пока не содержит приложений:

- папка `Web_1/` — корневой каталог (папка) нашего проекта PyCharm. Она представляет собой контейнер, в котором будут создаваться наши Django-проекты, приложения

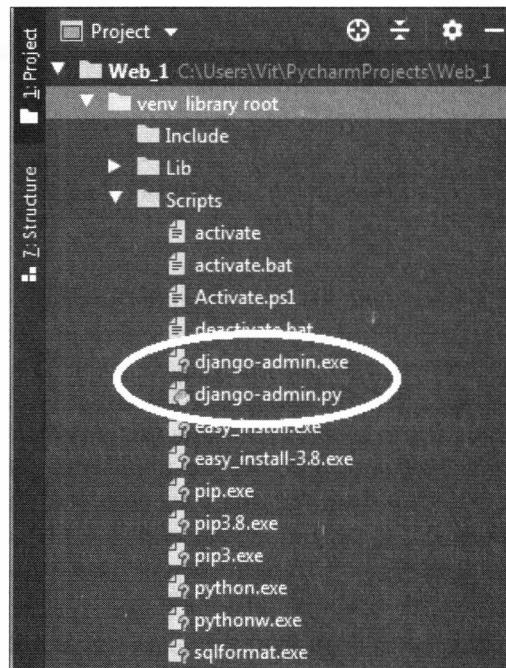


Рис. 4.2. Файлы Django в каталоге Script созданного проекта

```
Terminal: Local +  
Microsoft Windows [Version 6.1.7601]  
(c) Корпорация Майкрософт 2013. Все права защищены.  
(venv) C:\Users\Vit\PycharmProjects\Web_1>django-admin startproject hello
```

Рис. 4.3. Создание нового проекта Django в окне терминала PyCharm

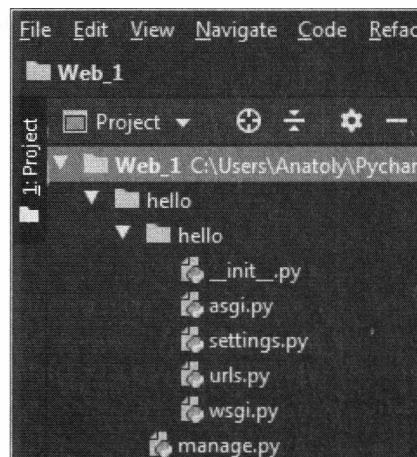


Рис. 4.4. Файловая структура проекта hello

и прочие файлы для управления проектом. Название папки ничего не значит для Django, и его можно поменять на свое усмотрение;

- папка `Web_1/hello` — это внешняя папка веб-проекта Django, которая содержит модуль для управления проектом и вложенную папку с аналогичным названием;
- файл `manage.py` — это менеджер проекта, или инструмент управления проектом из командной строки (в частности, при помощи этого менеджера можно запустить проект на исполнение на локальном веб-сервере Django);
- папка `Web_1/hello/hello/` — внутренняя папка проекта Django. Она содержит текущий и единственный на данный момент пакет (или, вернее, проект).



Рис. 4.5. Структура проекта Django с именем `hello`

Теперь обратимся к содержимому вложенной папки нашего первого проекта, а именно `Web_1/hello/hello/`:

- `hello/__init__.py` — пустой файл, предназначенный для регистрации пакетов. Наличие данного файла в каталоге `Web_1/` указывает интерпретатору Python на то, что в текущей папке находится пакет программных модулей;
- `hello/asgi.py` — точка входа для ASGI-совместимых веб-серверов, обслуживающих ваш проект (он потребуется при развертывании приложения на публичном сайте);
- `hello/settings.py` — файл для конфигурации текущего проекта Django;
- `hello/urls.py` — это URL-декларации текущего проекта Django, или, иначе говоря, это «диспетчер URL-адресов» Django-проекта;
- `hello/wsgi.py` — точки входа для WSGI-совместимого веб-сервера (данний файл потребуется при развертывании приложения на публичном сайте).

Проект Django, состоит из одного или нескольких отдельных приложений. Каждое приложение представляет какую-то определенную функциональность или группу

функций. Один проект может включать множество приложений. Таким образом, большой проект можно разбить на отдельные приложения и разрабатывать их относительно независимо друг от друга. Кроме того, это позволит затем переносить некоторые приложения из одного проекта в другой, тем самым сокращая сроки разработки проектов.

Пока мы создали только проект Django, в котором еще нет ни одного приложения. Несмотря на это, мы уже можем выполнить контрольный запуск проекта. Сначала нужно войти в папку `hello`, для чего в окне терминала PyCharm выполнить следующую команду:

```
cd hello
```

В результате этого будет выполнен переход из корневой папки `Web_1` в папку `hello` проекта Django (рис. 4.6).

Terminal: Local +

Microsoft Windows [Version 6.1.7601]  
(c) Корпорация Майкрософт 2013. Все права защищены.

```
(venv) C:\Users\Vit\PycharmProjects\Web_1>django-admin startproject hello
(venv) C:\Users\Vit\PycharmProjects\Web_1>cd hello
(venv) C:\Users\Vit\PycharmProjects\Web_1\hello>
```

Рис. 4.6. Переход в папку `hello` проекта Django

Теперь, находясь в этой папке, можно запустить наш проект на выполнение. Наберите в окне терминала команду запуска локального веб-сервера:

```
python manage.py runserver
```

В результате выполнения этой команды будет запущен локальный веб-сервер разработки на вашем компьютере с адресом <http://127.0.0.1:8000/> (рис. 4.7).

Terminal: Local +

```
(venv) C:\Users\Vit\PycharmProjects\Web_1\hello>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until
Run 'python manage.py migrate' to apply them.
July 08, 2020 - 09:07:18
Django version 3.0.8, using settings 'hello.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Рис. 4.7. Старт локального веб-сервера разработки

### ПРИМЕЧАНИЕ

Если после выполнения этой команды вы получили сообщение об ошибке:

```
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xcf in position 5:  
invalid continuation byte
```

то это означает, что в имени вашего компьютера присутствуют символы кириллицы — например, **Вит-ПК**. В таком случае нужно войти в свойства компьютера и поменять его имя (убрать символы ПК). Для этого следует щелкнуть правой кнопкой мыши на значке **Мой компьютер**, в открывшемся меню активировать опцию **Свойства** и выбрать левой кнопкой мыши ссылку **Изменить параметры** (рис. 4.8).

В открывшемся окне **Свойства системы** на вкладке **Имя компьютера** нажмите кнопку **Изменить** — откроется панель **Изменение имени компьютера или домена**, в которой можно убрать символы русского алфавита из имени компьютера. В данном случае имя компьютера **Вит-ПК** было заменено на **Вит** (рис. 4.9).

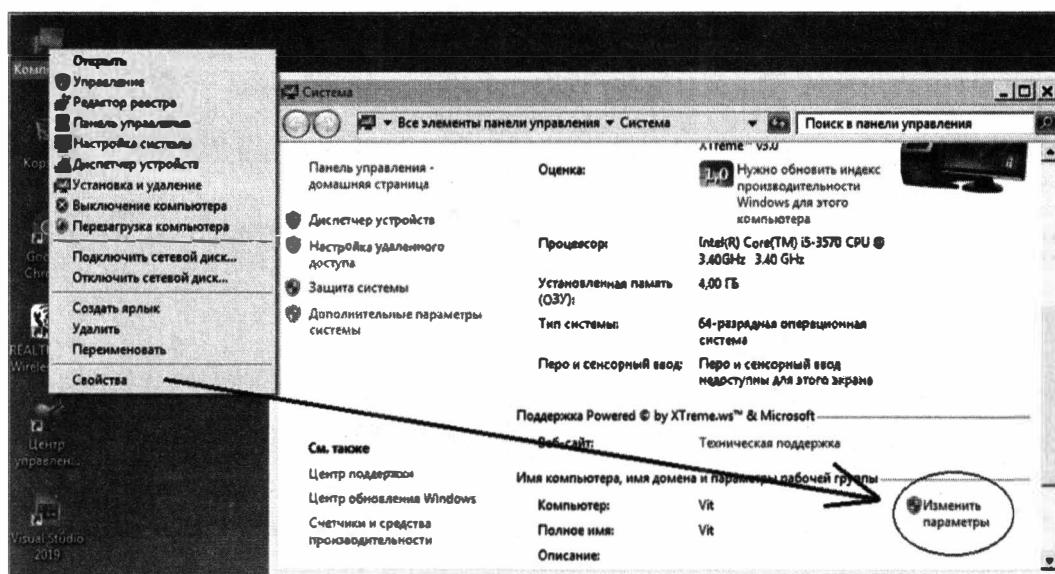


Рис. 4.8. Окно входа в режим изменения параметров компьютера

Команда `runserver` запускает сервер разработки Django, это облегченный веб-сервер, написанный исключительно на Python. Он включен в Django для того, чтобы можно было быстро приступить к работе над проектом, не занимаясь настройкой промышленного сервера типа Apache, до тех пор, пока веб-приложение будет отлажено и готово к работе в Интернете.

Если локальный сервер запустился без ошибок, щелкните левой кнопкой мыши на его ссылке: <http://127.0.0.1:8000/> (рис. 4.10).

В результате на вашем компьютере откроется веб-браузер, и в него будет загружена веб-страница поздравления с успешной установкой Django (рис. 4.11).

По умолчанию страница загружается на английском языке. Однако Django имеет локализацию и может работать с разными языками.

Остановите локальный веб-сервер нажатием комбинации клавиш **<Ctrl>+<C>**. Откройте файл `settings.py` и найдите фрагмент кода с установкой языка:

```
# Internationalization
# https://docs.djangoproject.com/en/3.0/topics/i18n/
LANGUAGE_CODE = 'en-us'
TIME_ZONE = 'UTC'
USE_I18N = True
USE_L10N = True
USE_TZ = True
```

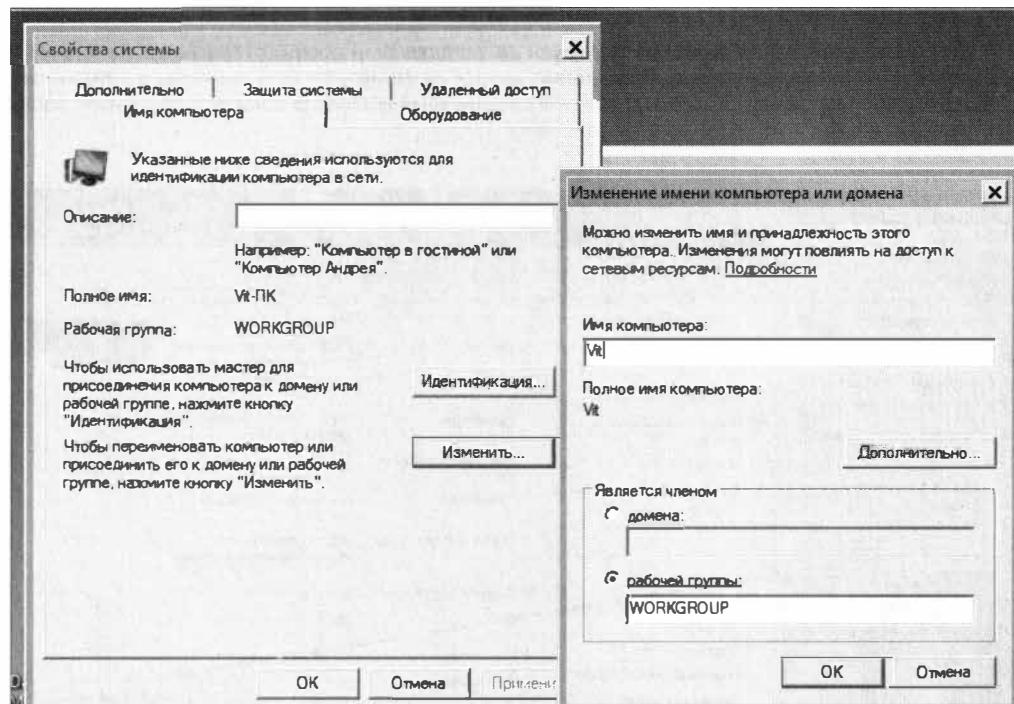


Рис. 4.9. Изменение имени компьютера

```
Terminal: Local +  

(venv) C:\Users\Vit\PycharmProjects\Web_1\hello>python ma  

Watching for file changes with StatReloader  

Performing system checks...  

System check identified no issues (0 silenced).  

You have 17 unapplied migration(s). Your project may not  

Run 'python manage.py migrate' to apply them.  

July 08, 2020 - 09:07:18  

Django version 3.0.8, using settings 'hello.settings'  

Starting development server at http://127.0.0.1:8000/  

Quit the server with CTRL-BREAK.
```

Рис. 4.10. Запуск локального веб-сервера разработчика

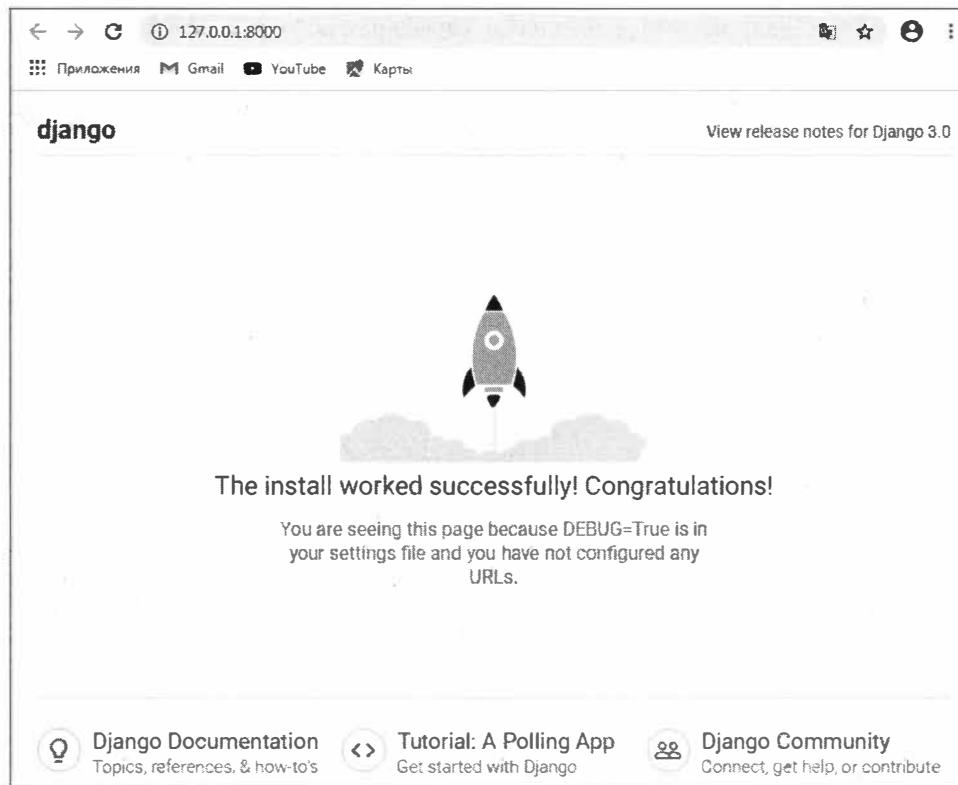


Рис. 4.11. Страница поздравления с успешной установкой Django

Как можно видеть, здесь по умолчанию задан английский язык в его американской версии (`en-us`). Задайте в этой опции русский язык (рис. 4.12):

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

```
settings.py ×

99     # Internationalization
100    # https://docs.djangoproject.com/en/3.0/topics/i18n/
101
102    LANGUAGE_CODE = 'ru-ru' # Red arrow points here
103
104    TIME_ZONE = 'UTC'
105
106    USE_I18N = True
107
108    USE_L10N = True
109
110    USE_TZ = True
```

Рис. 4.12. Задание русского языка в установках Django

Снова в окне терминала запустите локальный сервер разработки:

```
python manage.py runserver
```

и вы увидите, что после этого изменения окно приветствия и поздравления с успешной установкой Django будет выдано на русском языке (рис. 4.13).

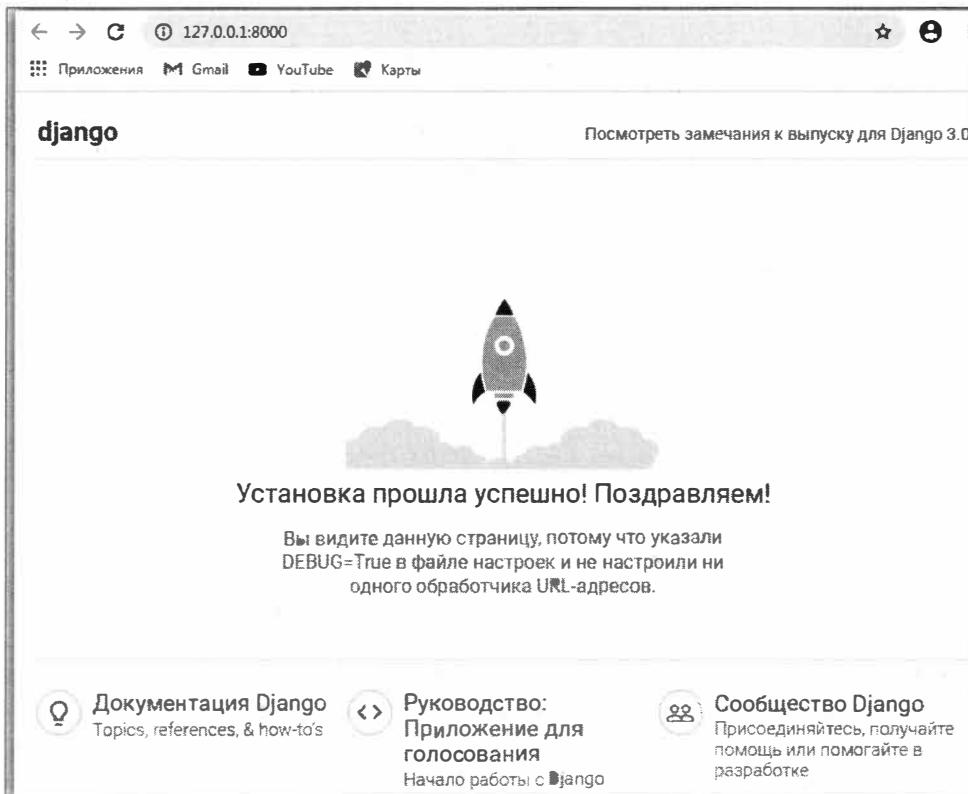


Рис. 4.13. Страница поздравления с успешной установкой Django на русском языке

Итак, нас можно поздравить с успешной установкой и пробным запуском Django!

Вернемся теперь к содержимому папки проекта `hello`. В ней появился файл `db.sqlite3`, это файл с базой данных SQLite (рис. 4.14).

SQLite — компактная встраиваемая реляционная база данных, исходный код которой передан в общественное достояние. Она является чисто реляционной базой данных.

Слово «встраиваемая» означает, что SQLite не использует парадигму «клиент–сервер». Иначе говоря, движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа. При этом в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном файле на том компьютере, на котором исполняется программа.

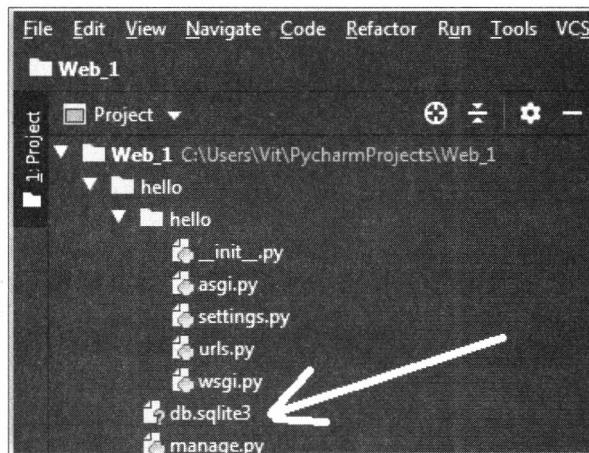


Рис. 4.14. Файл базы данных SQLite

В Django база данных SQLite создается автоматически (по умолчанию), формирование именно этой базы данных прописывается в файле конфигурации проекта. Если снова открыть файл `settings.py`, то в нем можно увидеть следующие строки:

```
# Database
# https://docs.djangoproject.com/en/3.0/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Именно здесь можно переопределить базу данных, с которой будет работать приложение. Чтобы работать с другими системами управления базами данных (СУБД), необходимо установить соответствующий дополнительный пакет (табл. 4.1).

Таблица 4.1. Основные базы данных, с которыми может работать приложение Django

СУБД	Пакет	Команда установки
PostgreSQL	psycopg2	pip install psycopg2
MySQL	mysql-python	pip install mysql-python
Oracle	cx_Oracle	pip install cx_Oracle

## 4.4. Первое приложение на Django

При создании проекта Django по умолчанию было сформировано несколько служебных приложений:

- django.contrib.admin;
- django.contrib.auth;

- django.contrib.contenttypes;
- django.contrib.sessions;
- django.contrib.messages;
- django.contrib.staticfiles.

Список всех приложений, созданных по умолчанию, можно найти в переменной `INSTALLED_APPS` файла проекта `settings.py`:

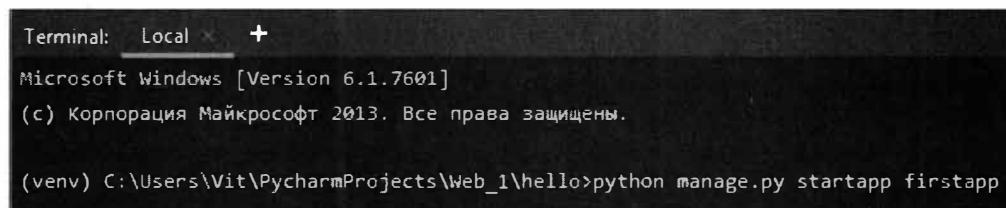
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Теперь у нас все готово для работы над своими приложениями, в которых мы сможем реализовать нужный нам функционал. Создадим в нашем проекте первое приложение, для которого затем напишем собственный программный код.

Для создания нового приложения в окне терминала PyCharm выполните следующую команду (рис. 4.15):

```
python manage.py startapp firstapp
```

Имя приложения может быть любым — оно указывается после команды `startapp`.



The screenshot shows a terminal window titled "Terminal: Local". The window displays the following text:  
 Microsoft Windows [Version 6.1.7601]  
 (c) Корпорация Майкрософт 2013. Все права защищены.  
 (venv) C:\Users\Vit\PycharmProjects\Web\_1\hello>python manage.py startapp firstapp

Рис. 4.15. Создание нового приложения `firstapp` в окне терминала PyCharm

В результате работы этой команды в проекте Django будет создано приложение `firstapp`, а в проекте появится новая папка, в которой будут храниться все файлы созданного приложения (рис. 4.16).

Рассмотрим структуру папок и файлов приложения `firstapp`:

- папка `migrations` — хранит информацию, позволяющую сопоставить базу данных со структурой данных, описанных в моделях;
- файл `__init__.py` — указывает интерпретатору Python, что текущий каталог будет рассматриваться в качестве пакета;
- файл `admin.py` — предназначен для административных функций (в частности, здесь выполняется регистрация моделей, которые используются в интерфейсе администратора);
- файл `apps.py` — определяет конфигурацию приложения;

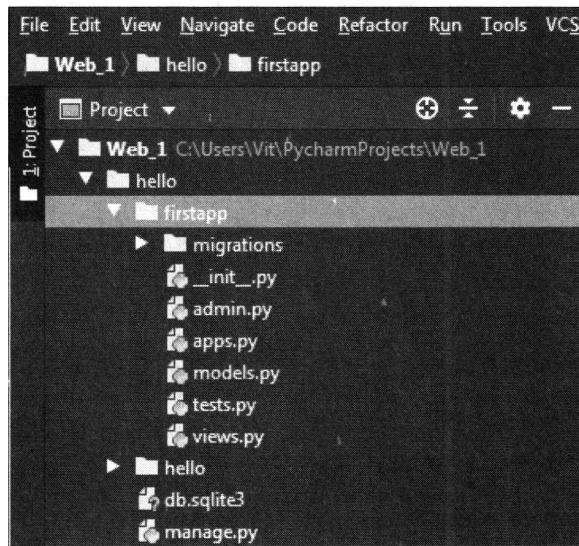


Рис. 4.16. Структура файлов приложения firstapp в окне терминала PyCharm

- файл `models.py` — хранит определение моделей, которые описывают данные, используемые в приложении;
- файл `tests.py` — хранит тесты приложения;
- файл `views.py` — определяет функции, которые получают запросы пользователей, обрабатывают их и возвращают ответ.

Но пока наше приложение никак не задействовано и его необходимо зарегистрировать в проекте Django. Для этого нужно открыть файл `settings.py` и добавить в конец массива с приложениями проекта `INSTALLED_APPS` наше приложение `firstapp` (добавляемый код выделен серым фоном):

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

В окне среды разработки PyCharm это будет выглядеть следующим образом (рис. 4.17).

В проекте может быть несколько приложений, и каждое из них нужно будет добавлять аналогичным образом. После добавления приложения `firstapp` полная структура нашего веб-приложения будет выглядеть так, как показано на рис. 4.18.

Теперь определим какие-нибудь простейшие действия, которые будет выполнять это приложение, — например, на запрос пользователя отправит ему в ответ приветствие **Hello World**. Напомним, что в запросе пользователь указывает адрес интересующей его HTML-страницы, а в ответ получает запрошенную страницу с неким содержимым, которое формируется сервером. В Django такое упрощенное взаимодействие можно

```

31     # Application definition
32
33     INSTALLED_APPS = [
34         'django.contrib.admin',
35         'django.contrib.auth',
36         'django.contrib.contenttypes',
37         'django.contrib.sessions',
38         'django.contrib.messages',
39         'django.contrib.staticfiles',
40         'firstapp',
41     ]

```

Рис. 4.17. Окно терминала PyCharm: регистрация приложения firstapp в файле settings.py

реализовать на уровне представления (view). Представление получает запрос, и оно же формирует ответ.

Для реализации этого функционала в приложении firstapp откроем файл views.py, который по умолчанию должен выглядеть так:

```
from django.shortcuts import render
# Create your views here.
```

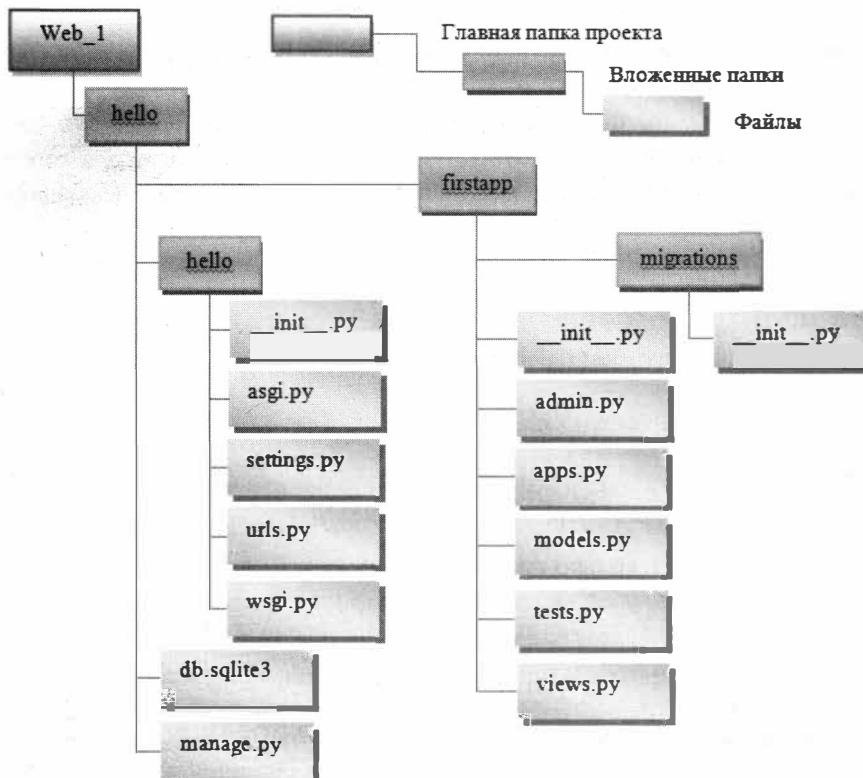


Рис. 4.18. Полная структура файлов и папок веб-приложения с проектом hello и приложением firstapp

Изменим этот код, как показано в листинге 4.1.

#### Листинг 4.1. Функция index

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
# Create your views here.
def index(request):
    return HttpResponseRedirect("Hello World! Это мой первый проект на Django!")
```

Здесь мы импортируем класс `HttpResponse` из стандартного пакета `django.http`. Затем определяем функцию `index()`, которая в качестве параметра получает объект `request` — запрос пользователя. Класс `HttpResponse` предназначен для создания ответа, который отправляется пользователю. Здесь с помощью кода `return HttpResponseRedirect` мы отправляем пользователю строку:

"Hello World! Это мой первый проект на Django!"

Это самое простое представление, которое можно реализовать в Django. Чтобы вызвать данное представление, нам нужно сопоставить его с URL-адресом, а для этого нам нужна конфигурация URL. Для того чтобы создать URLconf, в каталоге `firstapp` создайте новый файл с именем `urls.py`. Каталог вашего приложения теперь должен выглядеть следующим образом:

```
firstapp
    migrations
        __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
```

Теперь в приложении `firstapp` откроем созданный файл `firstapp/urls.py` и напишем в нем код, приведенный в листинге 4.2.

#### Листинг 4.2. Файл urls.py приложения firstapp

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
```

Чтобы использовать функцию `views.index`, в листинге 4.2 вначале выполнен импорт модуля `views`. Затем маршрут `('')` — это, по сути, запрос пользователя к корневой странице сайта, сопоставлен с функцией `views.index`, а также дополнительно задано имя для этого маршрута (`name='index'`). В результате, если пользователь запросит показать главную (корневую) страницу сайта `('')`, то будет вызвана функция `index` из файла `views.py`.

Следующий шаг — указание корневого URLconf. Для этого открываем файл `hello/urls.py`, и пишем в нем код, приведенный в листинге 4.3.

#### Листинг 4.3. Файл urls.py проекта hello

```
from django.contrib import admin
from django.urls import include, path
urlpatterns = [
    path('', include('firstapp.urls')),
    path('admin/', admin.site.urls),
]
```

В первой строке из модуля `django.contrib` импортируется класс `admin`, который обеспечивает работу с интерфейсом администратора сайта. Во второй строке из модуля `django.urls` импортируются функции `include` и `path`. Они дают возможность сопоставлять запросы пользователя (определенные маршруты) с функциями их обработки. Так, в нашем случае маршрут `'admin/'` будет обрабатываться методом `admin.site.urls`. Если пользователь запросит показать страницу администратора сайта (`'admin/'`), то будет вызван метод `admin.site.urls`, который вернет пользователю HTML-страницу администратора.

Функция `include()` позволяет ссылаться на другие URLconfs. Всякий раз, когда Django встречает `include()`, он отсекает любую часть URL-адреса, совпадающую с этой точкой, и отправляет оставшуюся строку во включенную конфигурацию URL для дальнейшей обработки.

Идея `include()` заключается в том, чтобы упростить URL-адреса plug-and-play. Поскольку приложение `firstapp` находится в собственной конфигурации URL (`firstapp/urls.py`), то URL-адреса можно поместить в `/firstapp/`, или в `/fun_firstapp/`, или в `/content/ firstapp /`, или в любой другой путь, и приложение по-прежнему будет работать.

Теперь снова запустим приложение командой

```
python manage.py runserver
```

И вновь перейдем в браузере по адресу `http://127.0.0.1:8000/`. После чего браузер нам отобразит строку: **Hello World! Это мой первый проект на Django!** (рис. 4.19).

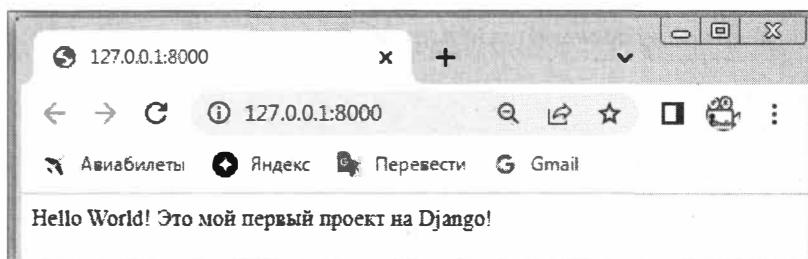


Рис. 4.19. Результаты работы приложения `firstapp`

Несколько слов о функции `path()`, которую мы использовали в листинге 4.3. Эта функция принимает четыре аргумента: два обязательных `route` и `view` и два необязательных `kwargs` и `name`. Пришло время рассмотреть, зачем нужны эти аргументы.

- **Аргумент route** (маршрут, путь) — это строка, содержащая шаблон URL. Когда Django обрабатывает запрос пользователя, он начинает перебор адресов с первого шаблона `urlpatterns` и продолжает спускаться вниз по списку, сравнивая запрошенный URL-адрес с каждым шаблоном, пока не найдет тот, который соответствует полученному от пользователя.

Обратите внимание, что эти регулярные выражения не ищут параметры `GET` и `POST` или имя домена. Например, в запросе к URL-адресу `https://www.example.com/myapp/` `URLconf` будет искать только `myapp/`.

В запросе к адресу `https://www.example.com/myapp/?page=3` `URLconf` также будет искать только `myapp/`.

- **Аргумент view.** Когда Django находит совпадение с регулярным выражением, он вызывает функцию, указанную в `view` с объектом `HttpRequest` в качестве первого аргумента и любыми «пойманными» значениями пути в качестве аргументов ключевого слова. Детально использование этого аргумента будет рассмотрено чуть позже на конкретных примерах.
- **Аргумент kwargs.** Описание данного аргумента мы пропустим, поскольку в примерах данной книги эта функциональность Django нам не понадобится.
- **Аргумент name.** Имя URL-адреса позволяет однозначно ссылаться на данный адрес из других частей Django, особенно из шаблонов. Это достаточно мощная и полезная функциональность, которая позволяет вносить глобальные изменения в шаблоны URL-адресов проекта, изменяя только один файл.

Поскольку маршрутизация запросов пользователя является ключевым моментом при разработке веб-приложений на Django, мы изучим маршрутизацию более подробно в следующей главе.

## 4.5. Краткие итоги

В этой главе были даны общие представления о Django и о структуре приложений на нем. Создан первый простейший проект на Django с выводом единственной страницы. Рассмотрены базовые сведения о представлениях (`view`) и маршрутизации запросов пользователей. Поскольку представления и маршрутизация запросов пользователей при разработке веб-приложений являются весьма важной частью Django, то эти вопросы более детально будут рассмотрены в следующей главе.





## ГЛАВА 5

# Представления и маршрутизация

Представления (*views*) и маршрутизация запросов пользователя — это ключевые элементы Django. Рассмотрению связанных с ними вопросов и посвящена эта глава, из материала которой вы узнаете:

- как обрабатываются запросы, поступившие от удаленного пользователя;
- маршруты, по которым обрабатываются запросы пользователей;
- регулярные выражения при описании маршрутов и их синтаксис;
- представления (*views*) и их параметры;
- параметры в строке запроса пользователя;
- переадресация запросов и отправка пользователю статусных кодов.

Многие разработчики, которые только осваивают веб-программирование, материал этой главы могут трудно воспринимать с первого прочтения. Однако не стоит отчаяваться, нужно просто сесть за компьютер, последовательно набрать и выполнить все приведенные здесь фрагменты программного кода. Это те основы, без понимания которых дальнейший материал книги просто не будет восприниматься. И это касается не только данной книги, а вообще перспектив освоения технологий разработки веб-приложений. Если же вам удастся осмыслить изложенный здесь материал и разобраться, каким образом обрабатываются запросы пользователей и как на основе этих запросов происходит адресация к тем или иным страницам сайта, то можно считать, что перед вами откроются двери в сложный, но весьма увлекательный мир веб-программирования.

### 5.1. Обработка запросов пользователей

Центральный момент любого веб-приложения — обработка запроса, который отправляет пользователь. В Django за обработку запроса отвечают *представления* (*views*). По сути, в представлениях реализованы функции обработки, которые принимают данные запроса пользователя в виде объекта `request` и генерируют некий ответ (`response`), который затем отправляется пользователю в виде HTML-страницы. По умолчанию представления размещаются в приложении в файле `views.py`.

Взглянем, например, на проект `hello`, в который добавлено приложение `firstapp`, созданное в предыдущей главе. При создании нового проекта файл `views.py` имел следующее содержимое:

```
from django.shortcuts import render
# Create your views here.
```

Этот код пока никак не обрабатывает запросы — он только импортирует функцию `render` (оказывать, предоставлять), которая может обрабатывать входящие запросы.

Генерировать результат обработки запроса (ответ пользователю) можно различными способами. Один из таких способов — использование класса `HttpResponse` (ответ HTTP) из пакета `django.http`, который позволяет отправить текстовое содержимое.

Изменим содержимое файла `views.py` приложения `firstapp`, как показано в листинге 5.1.

#### Листинг 5.1. Код файла `views.py`

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("<h2>Главная</h2>")
def about(request):
    return HttpResponse("<h2>О сайте</h2>")
def contact(request):
    return HttpResponse("<h2>Контакты</h2>")
```

В этом коде мы, по сути, запрограммировали функции для выдачи заголовков трех страниц сайта: **Главная (index)**, **О сайте (about)**, **Контакты (contact)**, как показано на рис. 5.1.

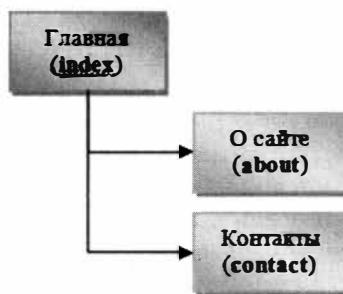


Рис. 5.1. Структура страниц сайта в приложении `firstapp`

В нашем случае здесь определены три функции, которые будут обрабатывать запросы пользователя. Каждая функция принимает в качестве параметра объект `request` (запрос). Для генерации ответа в конструкторе объекта `HttpResponse` (ответ HTTP) имеется некоторая строка. В данном случае ответом является строка кода HTML.

Чтобы эти функции сопоставлялись с запросами пользователя, нужно в файле `firstapp/urls.py` проекта `firstapp` определить для них маршруты. В частности, изменим этот файл, как показано в листинге 5.2.

**Листинг 5.2. Код файла firstapp/urls.py**

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index),
    path('about', views.about),
    path('contact', views.contact),
]
```

Переменная `urlpatterns` (шаблон URL) определяет набор сопоставлений запросов пользователя с функциями обработки этих запросов. В нашем случае, например, запрос пользователя к корню веб-сайта ('') будет обрабатываться функцией `index`, запрос по адресу 'about' — функцией `about`, а запрос 'contact' — функцией `contact`.

После этих изменений снова запустим приложение командой:

```
python manage.py runserver
```

и перейдем в браузере по адресу <http://127.0.0.1:8000/>. Браузер нам отобразит главную страницу сайта (рис. 5.2).

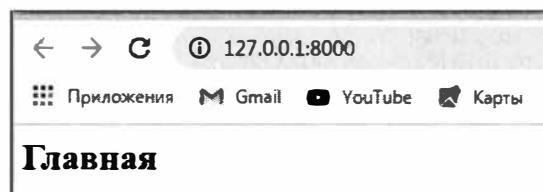


Рис. 5.2. Главная страница сайта

Если в адресной строке браузера изменить адрес на <http://127.0.0.1:8000/about>, то мы попадем на страницу **О сайте** (рис. 5.3).

Если же в адресной строке браузера изменить адрес на <http://127.0.0.1:8000/contact>, то мы попадем на страницу **Контакты** (рис. 5.4).

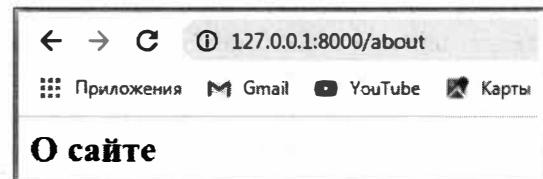


Рис. 5.3. Страница сайта О сайте

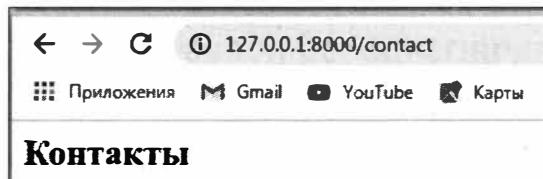


Рис. 5.4. Страница сайта Контакты

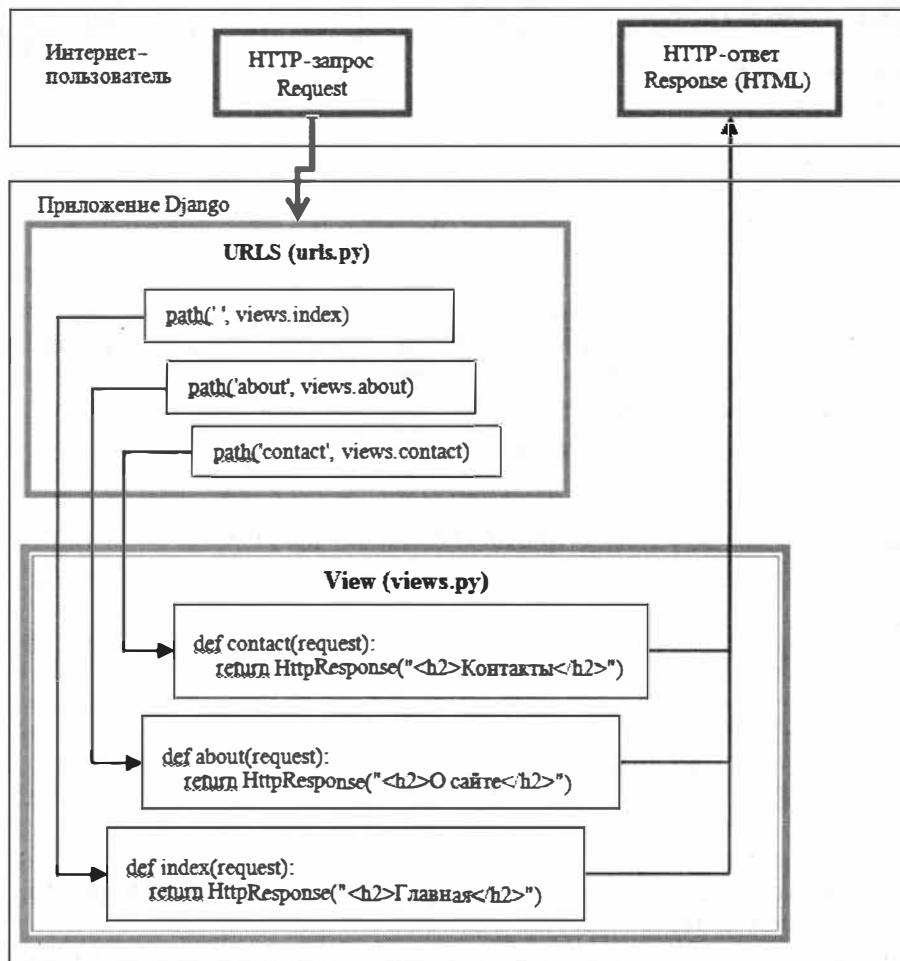


Рис. 5.5. Схема маршрутизации запросов пользователя и формирования ответов от веб-приложения

Схематично маршрутизация запросов пользователя и формирование ответов от веб-приложения представлена на рис. 5.5.

Следует отметить, что функция `path` при маршрутизации запросов пользователя при неточном описании параметров запроса может вызывать ошибки. В следующем разделе будет показано, как можно минимизировать ошибки при маршрутизации запросов пользователя.

## 5.2. Маршрутизация запросов пользователей в функциях `path` и `re_path`

В предыдущем разделе было рассмотрено сопоставление интернет-адресов (URL) и функций, которые обрабатывают запросы пользователей по этим адресам. В частности, мы создали в файле `firstapp/views.py` следующие функции (листинг 5.3).

**Листинг 5.3. Код файла firstapp/views.py**

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("<h2>Главная</h2>")
def about(request):
    return HttpResponse("<h2>О сайте</h2>")
def contact(request):
    return HttpResponse("<h2>Контакты</h2>")
```

А в файле `firstapp/urls.py` они сопоставляются с адресами URL с помощью функции `path()` (листинг 5.4).

**Листинг 5.4. Код файла firstapp/urls.py**

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index),
    path('about', views.about),
    path('contact', views.contact),
]
```

Здесь мы воспользовались функцией `path()`, расположенной в пакете `django.urls` и принимающей два параметра: запрошенный адрес URL и функцию, которая обрабатывает запрос по этому адресу. Дополнительно через третий параметр можно указать имя маршрута:

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

Здесь маршруту задано имя 'home'.

Однако функция `path` имеет серьезное ограничение — запрошенный путь должен абсолютно точно соответствовать указанному в маршруте адресу URL. Так, в приведенном ранее примере, функция `views.about` сможет корректно обработать запрос только в том случае, когда адрес будет в точности соответствовать значению 'about'. Например, стоит добавить к этому значению слеш ('about/'), и Django уже не сможет сопоставить путь с этим запросом, а мы в ответ получим ошибку (рис. 5.6).

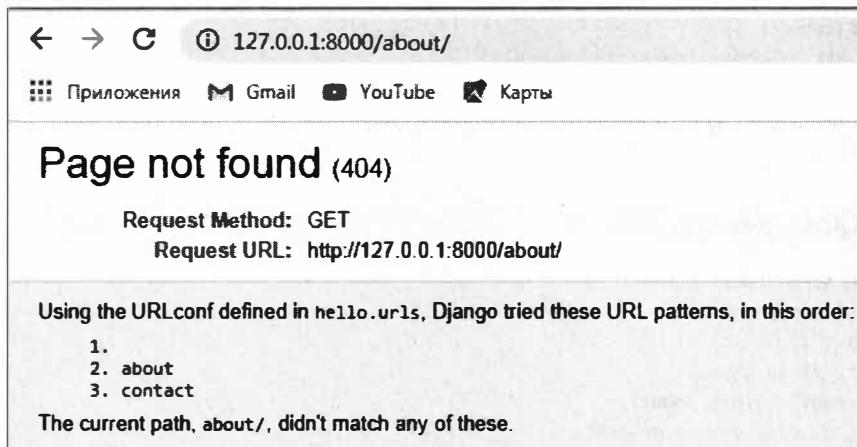
В качестве альтернативы для определения маршрутов мы можем вызвать функцию `re_path()`, также расположенную в пакете `django.urls`. Ее преимущество состоит в том, что она позволяет задать адреса URL с помощью *регулярных выражений*.

В качестве примера изменим содержание файла `firstapp/urls.py`, как показано в листинге 5.5.

**Листинг 5.5. Код файла firstapp/urls.py**

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

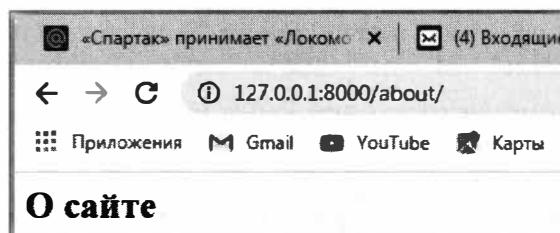
```
urlpatterns = [
    path('', views.index),
    re_path(r'^about', views.about),
    re_path(r'^contact', views.contact),
]
```



**Рис. 5.6.** Возврат ошибки при обращении к странице `about` по адресу `http://127.0.0.1:8000/about/`

Адрес в первом маршруте по-прежнему образуется с помощью функции `path` и указывает на «корень» веб-приложения. А остальные два маршрута образуются с помощью функций `re_path()`. Причем, поскольку определяется регулярное выражение, то перед строкой с шаблоном адреса URL ставится буква `r`. В самом шаблоне адреса возможны различные элементы синтаксиса регулярных выражений. В частности, выражение `^about` указывает на то, что адрес должен начинаться с `about`.

Однако он необязательно в точности должен соответствовать строке `'about'`, как это было в случае с функцией `path`. Мы можем обратиться по любому адресу — главное, чтобы он начинался с `about`, и тогда подобный запрос будет корректно обрабатываться функцией `views.about`. Если мы теперь укажем слеш в конце: `('about/')`, то Django корректно сопоставит путь с этим запросом, и мы в ответ получим запрошенную страницу (рис. 5.7).



**Рис. 5.7.** Корректное обращение к странице `about` по адресу `http://127.0.0.1:8000/about/` при использовании функции `re_path()`

## 5.3. Очередность маршрутов

Когда к приложению приходит запрос от пользователя, система проверяет соответствие запроса маршрутам по мере их следования в модуле `firstapp/urls.py`. Вначале сравнивается первый маршрут, если он не подходит, то сравнивается второй и т. д. Поэтому более общие маршруты должны определяться в последнюю очередь, а более конкретные маршруты — располагаться в начале этого модуля. Рассмотрим следующий пример следования маршрутов в переменной `urlpatterns`:

```
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    path('', views.index),
]
```

Здесь адрес `'^about/contact/'` представляет собой более конкретный маршрут по сравнению с `'^about'`. Поэтому он определяется в первую очередь.

## 5.4. Основные элементы синтаксиса регулярных выражений

В предыдущем разделе у нас был только один символ в регулярном выражении, определяющем начало адреса (^). Далее приведены некоторые базовые элементы регулярных выражений, которые пригодны для определения адресов URL:

- ^ — начало адреса;
- \$ — конец адреса;
- + — один и более символов;
- ? — ноль или один символ;
- {n} — n символов;
- {n, m} — от n до m символов;
- . — любой символ;
- \d+ — одна или несколько цифр;
- \D+ — одна или несколько НЕ цифр;
- \w+ — один или несколько буквенных символов.

В табл. 5.1 представлено несколько возможных сопоставлений шаблонов адресов и запросов.

**Таблица 5.1. Варианты возможных сопоставлений шаблонов адресов и запросов**

Адрес	Запрос
<code>r'^\$'</code>	<code>http://127.0.0.1/</code> (корень сайта)
<code>r'^about'</code>	<code>http://127.0.0.1/about/</code> или <code>http://127.0.0.1/about/contact</code>
<code>r'^about\contact'</code>	<code>http://127.0.0.1/about/contact</code>

Таблица 5.1 (окончание)

Адрес	Запрос
r'^products/\d+/'	<b>http://127.0.0.1/products/23/ или http://127.0.0.1/products/6459/abc</b> Не соответствует запросу: <b>http://127.0.0.1/products/abc/</b>
r'^products/\D+/'	<b>http://127.0.0.1/products/abc/ или http://127.0.0.1/products/abc/123</b> Не соответствует запросам: <b>http://127.0.0.1/products/123/ или http://127.0.0.1/products/123/abc</b>
r'^products/phones tablets/'	<b>http://127.0.0.1/products/phones/1 или http://127.0.0.1/products/tablets/</b> Не соответствует запросу: <b>http://127.0.0.1/products/clothes/</b>
r'^products/\w+/'	<b>http://127.0.0.1/abc/ или http://127.0.0.1/123/</b> Не соответствует запросу: <b>http://127.0.0.1/abc-123</b>
r'^products/[-\w]+/'	<b>http://127.0.0.1/abc-123</b>
r'^products/[A-Z]{2}/'	<b>http://127.0.0.1/RU</b> Не соответствует запросам: <b>http://127.0.0.1/Ru или http://127.0.0.1/RUS</b>

## 5.5. Параметры представлений

Функции-представления могут принимать параметры, через которые можно передавать различные данные. Такие параметры передаются в адресе URL. Например, в запросе <http://localhost/index/3/5/> последние два сегмента (`3/5/`) могут представлять параметры URL, которые можно связать с параметрами функции-представления через систему маршрутизации.

### 5.5.1. Определение параметров через функцию `re_path()`

Вернемся к нашему приложению `firstapp`, которое мы создали в предыдущей главе, и определим в этом приложении в файле `firstapp/views.py` дополнительные функции, приведенные в листинге 5.6.

#### Листинг 5.6. Код файла `firstapp/views.py`

```
def products(request, productid):
    output = "<h2>Продукт № {0}</h2>".format(productid)
    return HttpResponse(output)
```

```
def users(request, id, name):
    output = "<h2>Пользователь</h2><h3>id:" \
             " {0} Имя:{1}</h3>".format(id, name)
    return HttpResponseRedirect(output)
```

Здесь функция `products` кроме параметра `request` принимает также параметр `productid` (идентификатор товара). Отправляемый пользователю ответ содержит значение этого параметра.

Функция `users` принимает два дополнительных параметра: `id` и `name` (идентификатор и имя пользователя). И подобным же образом эти данные отправляются обратно пользователю.

Теперь изменим файл `urls.py`, чтобы он мог сопоставить эти функции с запросами пользователя (листинг 5.7, изменения выделены серым фоном).

#### Листинг 5.7. Код файла `firstapp/urls.py`

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

urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    re_path(r'^products/(?P<productid>\d+)/', views.products),
    re_path(r'^users/(?P<id>\d+)/(?P<name>\D+)/', views.users),
    path('', views.index),
]
```

Для представления параметра в шаблоне адреса используется выражение `?P<>`. Общее определение параметра соответствует формату:

`(?P<имя_параметра>регулярное_выражение)`

В угловых скобках помещается название параметра. После закрывающей угловой скобки следует регулярное выражение, которому должно соответствовать значение параметра.

Например, фрагмент: `?P<productid>\d+` определяет, что параметр называется `productid`, и он должен соответствовать регулярному выражению `\d+`, т. е. представлять последовательность цифр.

Во втором шаблоне адреса: `(?P<id>\d+)/(?P<name>\D+)` определяются два параметра: `id` и `name`. При этом параметр `id` должен представлять число, а параметр `name` — состоять только из буквенных символов.

Отметим также, что количество и название параметров в шаблонах адресов URL соответствуют количеству и названиям параметров функций, которые обрабатывают запросы по этим адресам.

Снова в окне терминала запустим локальный сервер разработки:

```
python manage.py runserver
```



Рис. 5.8. Главная страница сайта, загруженная по умолчанию

По умолчанию будет загружена главная страница сайта (рис. 5.8).

Теперь мы можем через адресную строку передать от пользователя данные в приложение в виде запроса. Наберем в адресной строке браузера следующий текст:

`http://127.0.0.1:8000/products/5`

Таким способом пользователь может запросить на веб-сайте информацию о продукте с идентификационным номером 5. В ответ пользователю будет возвращена страница с информацией о продукте № 5 (рис. 5.9).

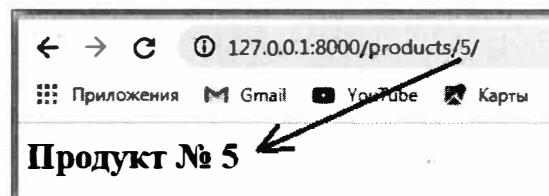


Рис. 5.9. Страница сайта, возвращенная пользователю, запросившему информацию о продукте № 5

Теперь попробуем через адресную строку сделать в приложение запрос о пользователе. Наберем в адресной строке браузера следующий текст:

`http://127.0.0.1:8000/users/3/Виктор/`

Так можно запросить на веб-сайте информацию о пользователе с идентификационным номером 3 по имени Виктор. В ответ будет возвращена страница с информацией о пользователе № 3 (рис. 5.10).

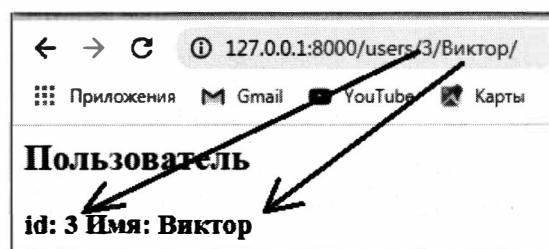


Рис. 5.10. Страница сайта, возвращенная пользователю, запросившему информацию о пользователе № 3

Однако если мы в запросе не укажем значение для параметра или передадим значение, которое не соответствует регулярному выражению, то система не сможет найти ресурс для обработки такого запроса и выдаст сообщение об ошибке. Например, если пользо-

ватель запросит информацию о продукте, но при этом не укажет идентификационный номер продукта, то система выдаст в ответ следующее сообщение об ошибке (рис. 5.11).

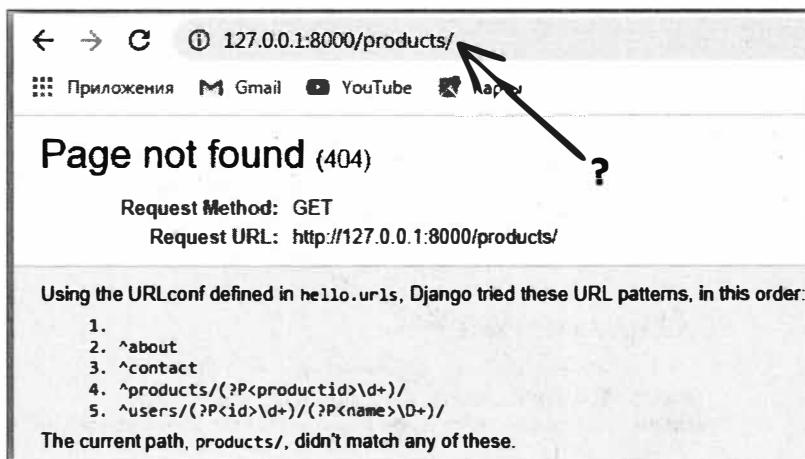


Рис. 5.11. Страница сайта, возвращенная пользователю, после ошибочного запроса информации о продукте

Для предотвращения ошибок такого рода можно в функции `products`, расположенной в файле `views.py`, определить значение параметра по умолчанию (листинг 5.8, изменения выделены серым фоном).

#### Листинг 5.8. Код файла firstapp/views.py

```
def products(request, productid = 1):
    output = "<h2>Продукт № {0}</h2>".format(productid)
    return HttpResponse(output)
```

Теперь, если в функцию не было передано значение для параметра `productid`, то он получает значение по умолчанию 1.

В этом случае в файле `urls.py` нужно дополнитель но определить еще один маршрут:

```
re_path(r'^products/$', views.products),
```

Добавив эту строчку, мы в итоге получим следующее содержание файла `urls.py` (листинг 5.9, изменения выделены серым фоном).

#### Листинг 5.9. Код файла firstapp/urls.py

```
from django.urls import path
from django.urls import re_path
from .import views
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
```

```

re_path(r'^products/(?P<productid>\d+)/', views.products),
re_path(r'^products/$', views.products), # маршрут по умолчанию
re_path(r'^users/(?P<id>\d+)/(?P<name>\D+)/', views.users),
path('', views.index),
]

```

Теперь если в запросе пользователя не будет указан `id` продукта, то система вернет ему не страницу с ошибкой, а страницу с номером продукта, который в функции `products` был задан по умолчанию: `productid=1` (рис. 5.12).

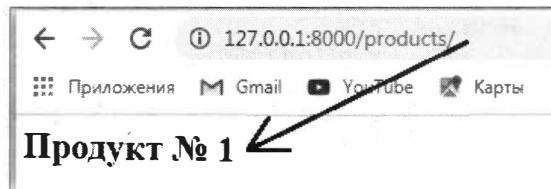


Рис. 5.12. Страница сайта, возвращенная пользователю, запросившему информацию о продукте без указания его идентификатора

## 5.5.2. Определение параметров через функцию `path()`

Вернемся к функциям, описанным в файле `firstapp/views.py` (листинг 5.10).

### Листинг 5.10. Код файла `firstapp/views.py`

```

from django.http import HttpResponse

def index(request):
    return HttpResponse("<h2>Главная</h2>")
def about(request):
    return HttpResponse("<h2>О сайте</h2>")
def contact(request):
    return HttpResponse("<h2>Контакты</h2>")
def products(request, productid=1):
    output = "<h2>Продукт № {0}</h2>".format(productid)
    return HttpResponse(output)
def users(request, id, name):
    output = "<h2>Пользователь</h2><h3>id:" \
             " {0} Имя:{1}</h3>".format(id, name)
    return HttpResponse(output)

```

Здесь функция `products` принимает параметр «идентификатор» продукта (`productid`), а функция `users` — идентификатор (`id`) и имя (`name`) пользователя. При этом передача в данные функции этих параметров в файле `firstapp/urls.py` выполняется с помощью функции `re_path()`. Откроем файл `firstapp/urls.py` и сменим в маршрутах адресации для них функцию `re_path()` на функцию `path()`, закомментировав соответствующие строки. Определение параметров в файле `firstapp/urls.py` с помощью функции `path()` будет выглядеть как в листинге 5.11 (изменения выделены серым фоном).

**Листинг 5.11. Код файла firstapp/urls.py**

```

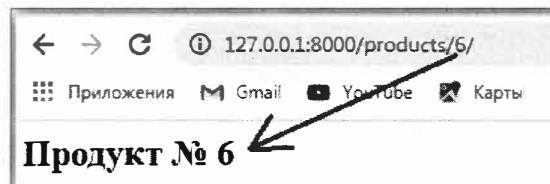
from django.urls import path
from django.urls import re_path
from . import views
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    # re_path(r'^products/(?P<productid>\d+)/', views.products),
    path('products/<int:productid>', views.products),
    re_path(r'^products/$', views.products), # маршрут по умолчанию
    # re_path(r'^users/(?P<id>\d+)/(?P<name>\D+)/', views.users),
    path('users/<int:id>/<name>', views.users),
    path('', views.index),
]

```

Проверим, корректно ли работает функция `path()` при запросе пользователя на получение информации о продукте. Для этого снова в окне терминала запустим локальный сервер разработки:

```
python manage.py runserver
```

и обратимся к странице `products` (рис. 5.13).



**Рис. 5.13.** Страница сайта, возвращенная пользователю, запросившему информацию о продукте, при использовании функции `path()` в файле `urls.py`

Как можно видеть, функция `path()` работает здесь аналогично функции `re_path`. В зависимости от предпочтений программиста можно использовать любую из этих функций.

Параметры функции `path()` заключаются в угловые скобки в формате:

<спецификатор: название\_параметра>

В нашем примере в маршруте обращения к странице с продуктами параметр `productid` имеет спецификатор `int` (целое число).

По умолчанию Django предоставляет следующие спецификаторы параметров функций:

- `str` — соответствует любой строке за исключением символа `(/)`. Если спецификатор не указан, то `str` используется по умолчанию;
- `int` — соответствует любому положительному целому числу;
- `slug` — соответствует последовательности буквенных символов ASCII, цифр, дефиса и символа подчеркивания, например: `building-your-1st-django-site`;
- `uuid` — соответствует идентификатору UUID, например: `075194d3-6885-417e-a8a8-6c931e272f00`;

- path — соответствует любой строке, которая также может включать символ (/), в отличие от спецификатора str.

### 5.5.3. Определение параметров по умолчанию в функции *path()*

Для примера назначим функциям в файле `views.py` значения параметров по умолчанию для тех страниц сайта, которые выдают информацию о продуктах и пользователях. Мы уже ранее задавали для продуктов значение по умолчанию: `productid=1`. Теперь укажем для пользователя значения по умолчанию идентификатора пользователя (`id=1`) и имени пользователя (`name="Максим"`). После таких изменений файл `firstapp/views.py` будет выглядеть как в листинге 5.12 (изменения выделены серым фоном).

#### Листинг 5.12. Код файла `firstapp/views.py`

```
from django.http import HttpResponseRedirect
def index(request):
    return HttpResponseRedirect("<h2>Главная</h2>")
def about(request):
    return HttpResponseRedirect("<h2>О сайте</h2>")
def contact(request):
    return HttpResponseRedirect("<h2>Контакты</h2>")
def products(request, productid=1):
    output = "<h2>Продукт № {0}</h2>".format(productid)
    return HttpResponseRedirect(output)
def users(request, id=1, name="Максим"):
    output = "<h2>Пользователь</h2><h3>id:" \
        " {0} Имя: {1}</h3>".format(id, name)
    return HttpResponseRedirect(output)
```

После этого для функций `products` и `users` в файле `firstapp/urls.py` с использованием функции `path()` нужно определить по два маршрута: для запроса с заданными параметрами и для запроса с параметрами по умолчанию (листинг 5.13).

#### Листинг 5.13. Код файла `firstapp/urls.py`

```
path('products/', views.products), # маршрут по умолчанию
path('products/<int:productid>', views.products),
path('users/', views.users), # маршрут по умолчанию
path('users/<int:id>/<str:name>', views.users),
```

После этих изменений файл `firstapp/urls.py` будет выглядеть как в листинге 5.14.

#### Листинг 5.14. Код файла `firstapp/urls.py`

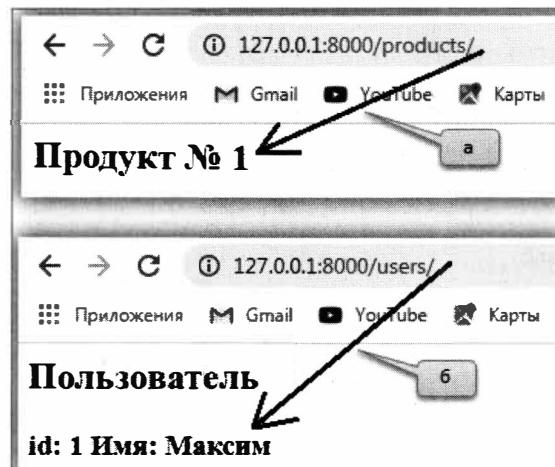
```
from django.urls import path
from django.urls import re_path
from . import views
```

```
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    path('products/', views.products), # маршрут по умолчанию
    path('products/<int:productid>/', views.products),
    path('users/', views.users), # маршрут по умолчанию
    path('users/<int:id>/<str:name>/', views.users),
    path('', views.index),
]
```

Проверим, корректно ли работает функция `path()` при запросе на получение информации о продукте и пользователе в том случае, когда в запросе пользователя указаны не все параметры продукта и пользователя. Для этого снова в окне терминала запустим локальный сервер разработки:

```
python manage.py runserver
```

и обратимся к страницам `products` и `users` (рис. 5.14).



**Рис. 5.14.** Страницы сайта, возвращенные пользователю, запросившему информацию о продукте (вверху) и пользователе (внизу), с параметрами по умолчанию при использовании функции `path()` в файле `urls.py`

Как можно видеть, при отсутствии параметров в запросе пользователя функция `path()` не выдает здесь ошибку, а возвращает ответные страницы с параметрами функций по умолчанию.

## 5.6. Параметры строки запроса пользователя

Следует четко различать параметры, которые передаются через *интернет-адрес (URL)*, и параметры, которые передаются через *строку запроса*. Например, в запросе:

`http://localhost/index/3/Виктор/`

два последних сегмента: `3/Виктор/` представляют собой параметры URL.

А вот в запросе:

**http://localhost/index?id=3&name= Виктор**

те же самые значения **3** и **Виктор** представляют собой параметры строки запроса.

Параметры строки запроса указываются после символа вопросительного знака (**?**). Каждый такой параметр представляет собой пару «ключ–значение». Например, в параметре **id=3**: **id** — это ключ параметра, а **3** — его значение. Параметры в строке запроса отделяются друг от друга знаком амперсанда (**&**).

Для получения параметров из строки запроса применяется метод `request.GET.get()`.

Например, переопределим в файле `firstapp/views.py` функции `products()` и `users()`, как показано в листинге 5.15 (изменения выделены серым фоном).

### Листинг 5.15. Код файла `firstapp/views.py`

```
from django.http import HttpResponse
def index(request):
    return HttpResponse("<h2>Главная</h2>")
def about(request):
    return HttpResponse("<h2>О сайте</h2>")
def contact(request):
    return HttpResponse("<h2>Контакты</h2>")
def products(request, productid=1):
    category = request.GET.get("cat", "Не задана")
    output = "<h2>Продукт № {0} Категория: {1}</h2>"\
        .format(productid, category)
    return HttpResponse(output)
def users(request):
    id = request.GET.get("id", "Не задано")
    name = request.GET.get("name", "Не задано")
    output = "<h2>Пользователь</h2> <h3>id: {0} Имя: {1}</h3 >"\
        .format(id, name)
    return HttpResponse(output)
```

Функция `products` принимает обычный параметр `productid` (идентификатор продукта), который будет передаваться через интернет-адрес (URL). И также из строки запроса извлекается значение параметра `cat` (категория продукта) — `request.GET.get("cat", "Не задана")`. Здесь первый аргумент функции — это название параметра строки запроса, значение которого нужно извлечь, а второй аргумент — значение по умолчанию (на случай, если в строке запроса не оказалось подобного параметра).

В функции `users` из строки запроса извлекаются значения параметров `id` и `name`. При этом заданы следующие значения параметров по умолчанию:

- `id= "Не задано";`
- `name= "Не задано".`

Снова в окне терминала запустим локальный сервер разработки:

`python manage.py runserver`

и обратимся к страницам `products` и `users`.

Сначала выполним обращение к странице `products` по следующим адресам:

- `http://127.0.0.1:8000/products/;`
- `http://127.0.0.1:8000/products/3;`
- `http://127.0.0.1:8000/products/3/?cat=Телефоны.`

В результате этих действий получим следующие результаты (рис. 5.15).

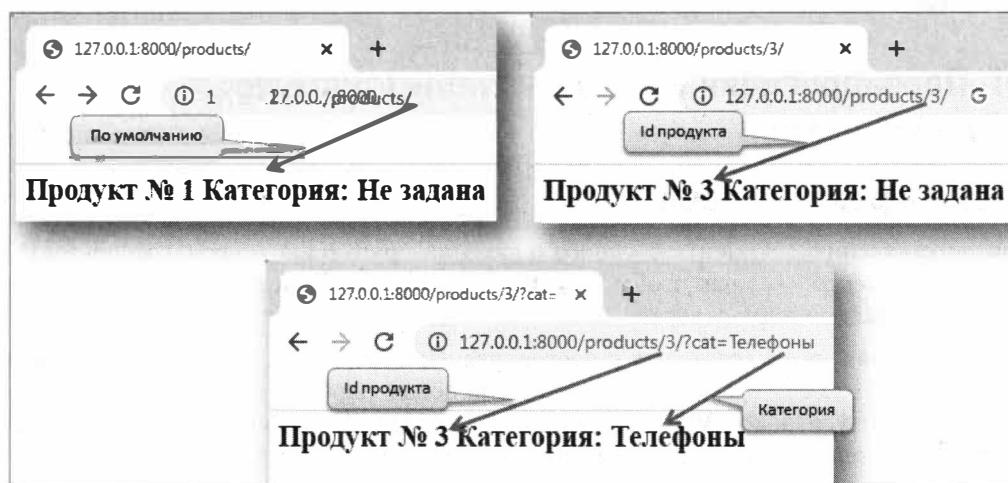


Рис. 5.15. Страница сайта, возвращенная пользователю, при разных параметрах запроса

Как видно из данного рисунка, при обращении к странице `products` без указания параметров вернулся ответ с параметрами по умолчанию. При указании в запросе числа 3 оно будет представлять фрагмент URL, присваиваемый параметру `product_id`, а значение `cat=Телефоны` — представлять параметр `cat` в строке запроса.

Теперь выполним обращение к странице `users` по следующим адресам:

- `http://127.0.0.1:8000/users/;`
- `http://127.0.0.1:8000/users/?id=8&name=Алексей.`

Выполнив эти действия, получим следующие результаты (рис. 5.16).

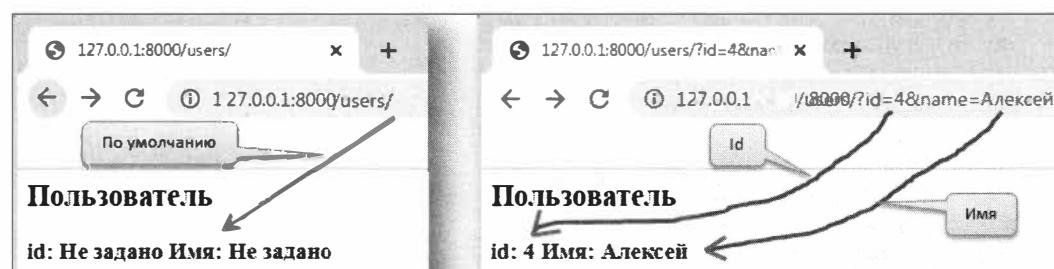


Рис. 5.16. Страница сайта, возвращенная пользователю, запросившему информацию о пользователе № 8 с именем Алексей

Как видно из данного рисунка, при обращении к странице `users` без указания параметров, вернулся ответ с параметрами по умолчанию. Во втором случае значения идентификатора — **4** и имя — **Алексей** переданы на страницу сайта через параметры `id` и `name`.

Итак, мы рассмотрели очень важные моменты, связанные с адресацией запросов пользователя и формирования для него ответных страниц. Но часто возникает потребность в переадресации, поскольку после модификации сайта данные могут быть перемещены по другому адресу. Перейдем теперь к рассмотрению способов решения вопроса переадресации.

## 5.7. Переадресация и отправка пользователю статусных кодов

### 5.7.1. Переадресация

При перемещении документа с одного адреса на другой мы можем воспользоваться механизмом *переадресации*, чтобы указать пользователям и поисковику, что документ теперь доступен по новому адресу.

Переадресация бывает временная и постоянная. При *временной* переадресации мы указываем, что документ временно перемещен на новый адрес. В этом случае в ответ отправляется статусный код **302**. При постоянной переадресации мы уведомляем систему, что документ теперь постоянно будет доступен по новому адресу.

Классы для создания переадресации:

- `HttpResponseRedirect` — временной;
- `HttpResponsePermanentRedirect` — постоянной.

Оба класса расположены в пакете `django.http`.

Для рассмотрения переадресации в качестве примера изменим код файла `firstapp/views.py` как показано в листинге 5.16 (изменения выделены серым фоном).

#### Листинг 5.16. Код файла `firstapp/views.py`

```
from django.http import HttpResponseRedirect, \
    HttpResponseRedirectPermanentRedirect, \
    HttpResponseRedirect

def index(request):
    return HttpResponseRedirect("<h2>Главная</h2>")
def about(request):
    return HttpResponseRedirect("<h2>О сайте</h2>")
def contact(request):
    return HttpResponseRedirectPermanentRedirect("/about")
#    return HttpResponseRedirect("<h2>Контакты</h2>")
def products(request, productid=1):
    category = request.GET.get("cat", "Не задана")
    output = "<h2>Продукт № {0} Категория: {1}</h2>"\
        .format(productid, category)
    return HttpResponseRedirect(output)
```

```
def users(request):
    id = request.GET.get("id", "Не задано")
    name = request.GET.get("name", "Не задано")
    output = "<h2>Пользователь</h2> <h3>id: {0} Имя: {1}</h3 >"\
        .format(id, name)
    return HttpResponse(output)
def details(request):
    return HttpResponseRedirect("/")
```

Что мы здесь сделали? При обращении к функции `contact` она станет перенаправлять пользователя по пути `"about"`, который будет обрабатываться функцией `about`. А функция `details` реализует постоянную переадресацию и перенаправит пользователя на «корень» (главную страницу) веб-приложения.

Для этого примера нам также необходимо в файле `firstapp/urls.py` добавить новый маршрут — `details` (листинг 5.17, изменения выделены серым фоном).

#### Листинг 5.17. Код файла `firstapp/urls.py`

```
from django.urls import path
from django.urls import re_path
from .import views
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    path('products/', views.products), # маршрут по умолчанию
    path('products/<int:productid>', views.products),
    path('users/', views.users), # маршрут по умолчанию
    path('users/<int:id>/<str:name>', views.users),
    path('details/', views.details),
    path('', views.index),
]
```

Для проверки работы переадресации в окне терминала запустим локальный сервер разработки:

```
python manage.py runserver
```

По умолчанию будет загружена главная страница **Index** (рис. 5.17).

Теперь в адресной строке браузера наберем адрес страницы **contact**:

**http://127.0.0.1:8000/contact**

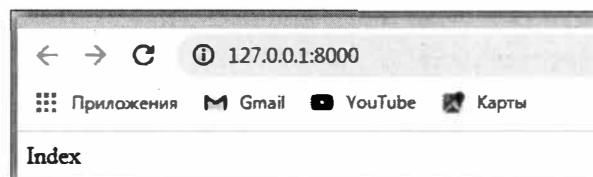


Рис. 5.17. Главная страница сайта, загруженная по умолчанию

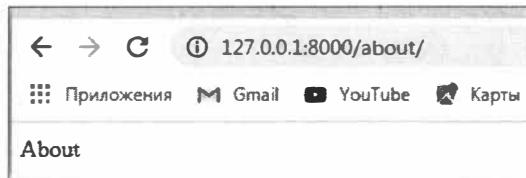


Рис. 5.18. Переадресация на страницу **About** при попытке вызова страницы **contact**

Поскольку мы задали переадресацию, вместо страницы **contact** будет загружена страница **About** (рис. 5.18).

Теперь в адресной строке браузера наберем адрес страницы **details**:

**http://127.0.0.1:8000/details**

Поскольку мы задали переадресацию, то вместо страницы **details** будет загружена главная страница сайта **Index** (рис. 5.19).

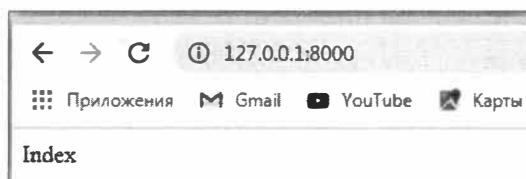


Рис. 5.19. Переадресация на страницу **Index** при попытке вызова страницы **details**

## 5.7.2. Отправка пользователю статусных кодов

В пакете `django.http` есть ряд классов, которые позволяют отправлять пользователю определенный *статусный код* (табл. 5.2).

Таблица 5.2. Перечень статусных кодов

Статусный код	Класс	Назначение кода
304 (Not Modified)	<code>HttpResponseNotModified</code>	Не модифицировать
400 (Bad Request)	<code>HttpResponseBadRequest</code>	Некорректный запрос
403 (Forbidden)	<code>HttpResponseForbidden</code>	Действия запрещены
404 (Not Found)	<code>HttpResponseNotFound</code>	Данные не найдены
405 (Method Not Allowed)	<code>HttpResponseNotAllowed</code>	Метод не разрешен
410 (Gone)	<code>HttpResponseGone</code>	Удаленный, недоступный
500 (Internal Server Error)	<code>HttpResponseServerError</code>	Внутренняя ошибка сервера

Вот пример применения этих классов:

```
from django.http import *
def m304(request):
    return HttpResponseRedirect()
def m400(request):
    return HttpResponseBadRequest("<h2>Bad Request</h2>")
```

```

def m403(request):
    return HttpResponseRedirect("Forbidden")
def m404(request):
    return HttpResponseRedirect("Not Found")
def m405(request):
    return HttpResponseRedirectNotAllowed("Method is not allowed")
def m410(request):
    return HttpResponseRedirectGone("Content is no longer here")
def m500(request):
    return HttpResponseRedirectServerError("Something is wrong")

```

Реализуем небольшой пример отправки пользователю статусных кодов. Откроем файл `views.py` и добавим в него новую функцию с именем `access` (листинг 5.18).

#### Листинг 5.18. Код файла `firstapp/views.py`

```

from django.http import HttpResponseRedirectBadRequest, HttpResponseRedirectForbidden
def access(request, age):
    # если возраст НЕ входит в диапазон 1-110, посыпаем ошибку 400
    if age not in range(1, 111):
        return HttpResponseRedirectBadRequest("Некорректные данные")
    if age > 17: # если возраст больше 17, то доступ разрешен
        return HttpResponseRedirect("Доступ разрешен")
    else: # если нет, то возвращаем ошибку 403
        return HttpResponseRedirectForbidden("Доступ заблокирован: недостаточно лет")

```

В данной функции в зависимости от возраста, полученного в запросе от пользователя, ему возвращается сообщение о статусе дальнейших действий. При этом возможны три ответа:

- «Некорректные данные» — если возраст находится вне диапазона от 1 до 110 лет;
- «Доступ разрешен» — если возраст превышает 17 лет (18 и более);
- «Доступ заблокирован: недостаточно лет» — если возраст менее 18 лет.

Для этого примера нам также необходимо в файле `firstapp/urls.py` добавить новый маршрут — `access` (листинг 5.19, изменения выделены серым фоном).

#### Листинг 5.19. Код файла `firstapp/urls.py`

```

from django.urls import path
from django.urls import re_path
from . import views
urlpatterns = [
    re_path(r'^contact/', views.contact),
    re_path(r'^about', views.about),
    path('products/', views.products), # маршрут по умолчанию
    path('products/<int:productid>', views.products),
    path('users/', views.users), # маршрут по умолчанию
    path('users/<int:id>/<str:name>', views.users),
    path('details/', views.details),
    path('', views.index),
    path("access/<int:age>", views.access),
]

```

Для проверки отправки статусных кодов в окне терминала запустим локальный сервер разработки:

```
python manage.py runserver
```

Теперь в адресной строке браузера наберем адрес страницы **access** и передадим в адресной строке запроса три разных возраста:

- /access/120;
- /access/25;
- /access/12.

В результате этих действий с сервера будут получены следующие ответы (рис. 5.20).

Как видно из данного рисунка, при разных значениях возраста, переданного в запросе пользователя, сервер вернул разные статусные ответы.

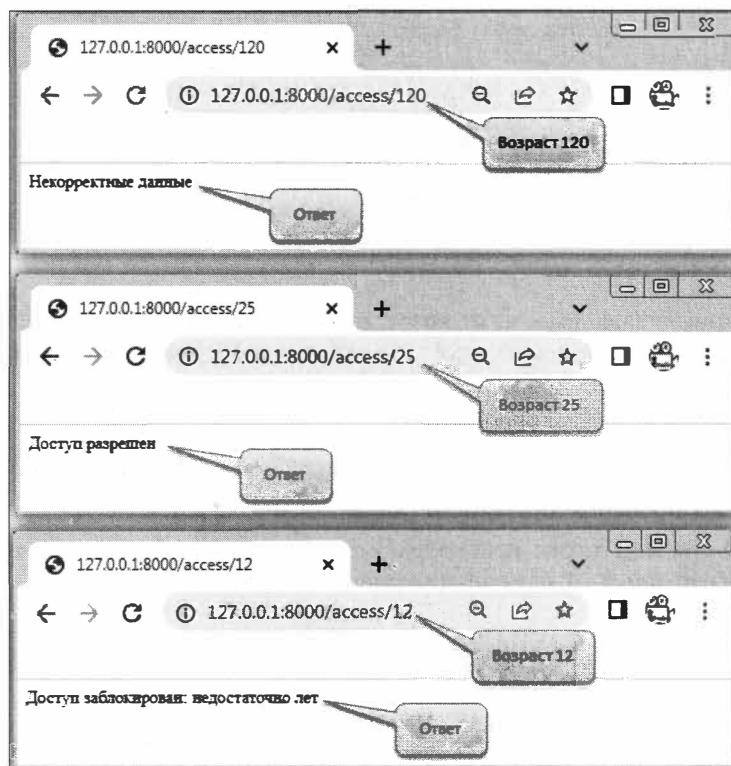


Рис. 5.20. Ответы сервера при разных значениях переданного возраста

## 5.8. Краткие итоги

В этой главе были представлены сведения о ключевых элементах Django: представлениях (views) и маршрутизации запросов пользователей. Ответы на запросы, поступившие от пользователя, возвращаются ему в виде HTML-страниц. В свою очередь, эти страницы формируются на основе шаблонов. Созданию и использованию шаблонов посвящена следующая глава.



## ГЛАВА 6

# Шаблоны в Django

Django отображает данные на сайте с помощью шаблонов и встроенных в шаблоны тегов. Шаблоны представляют собой HTML-страницы. Дело в том, что на страницах HTML нельзя в прямую размещать код Python, поскольку браузеры его не воспринимают — они понимают только HTML. Мы также знаем, что страницы HTML по своей сути статичны, в то время как Python позволяет вносить в программы динамические изменения. Так вот, теги шаблонов Django позволяют вставлять результаты работы программы на Python в HTML-страницы, что дает нам возможность создавать динамические веб-сайты быстрее и проще. Из этой главы мы узнаем:

- что такое шаблоны и для чего они нужны;
- как использовать функцию `render()` и класс `TemplateResponse` для загрузки шаблонов;
- как передать в шаблоны простые и сложные данные;
- что такое статичные файлы, файлы CSS и как использовать статичные файлы в приложениях на Django;
- как можно изменить конфигурацию шаблонов HTML-страниц;
- как расширить шаблоны HTML-страниц на основе базового шаблона;
- как можно применять специальные теги в шаблонах HTML-страниц.

### 6.1. Создание простейшего шаблона

Шаблоны (templates) отвечают за формирование внешнего вида приложения. Они предоставляют специальный синтаксис, который позволяет внедрять данные в код HTML.

В предыдущих главах мы создали и доработали проект `hello` с приложением `firstapp`. Теперь добавим в проект шаблоны. Для этого определим в корневой папке проекта новый каталог с именем `templates` (рис. 6.1). Вообще-то имя папки с шаблонами может быть любым, но, как правило, для лучшего восприятия структуры проекта этой папке все же лучше присвоить значащее имя `templates`.

Теперь нам нужно указать, что этот каталог будет служить хранилищем шаблонов. Для этого откроем файл `settings.py`. Шаблоны в нем настраивают с помощью переменной `TEMPLATES`:

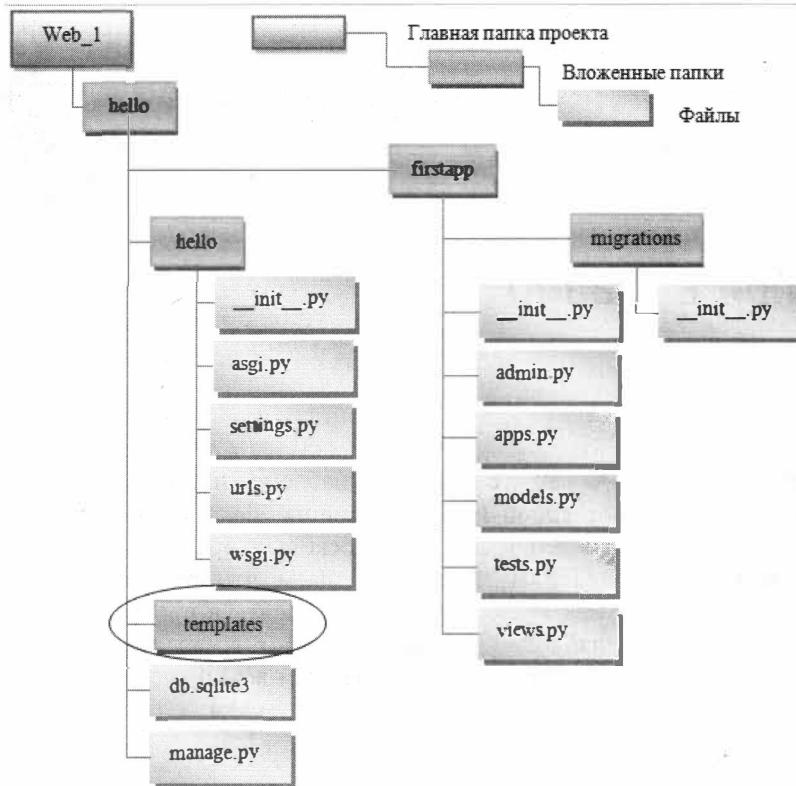


Рис. 6.1. Добавление папки templates в веб-приложение с проектом firstapp

```
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [],
    'APP_DIRS': True,
    'OPTIONS': {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
            'django.contrib.auth.context_processors.auth',
            'django.contrib.messages.context_processors.messages',
        ],
    },
},
]
```

Здесь задействованы следующие переменные:

- BACKEND — указывает, что нужно использовать шаблоны Django;
- DIRS — указывает на каталоги (папки) в проекте, которые будут содержать шаблоны;
- APP\_DIRS — при значении True указывает, что поиск шаблонов будет осуществляться не только в каталогах (папках), указанных в параметре DIRS, но и в дочерних катало-

гах (вложенных папках). Если такое поведение недопустимо, то можно установить значение `False`;

- `OPTIONS` — указывает, какие обработчики (процессоры) будут использоваться при обработке шаблонов.

Здесь параметр `DIRS` задает набор каталогов, которые хранят шаблоны. Но по умолчанию он пуст. Изменим этот фрагмент кода как показано в листинге 6.1 (изменения выделены серым фоном).

#### Листинг 6.1. Фрагмент файла `settings.py`

```
import os
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR,],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

Внесенные в проект изменения показаны на рис. 6.2.

Итак, мы определились с папкой, в которой будут находиться шаблоны, — это папка `hello/templates`. Теперь можно создать наш первый шаблон, в папке `templates` создадим новый файл с именем `index.html`. Для этого в окне программной оболочки PyCharm нужно щелкнуть правой кнопкой мыши на имени проекта `Web_1` и из появившегося меню выбрать опцию: `template ->File ->HTML File` (рис. 6.3). После этого откроется окно `New HTML File` (рис. 6.4), в котором введите имя создаваемого файла `index` и нажмите клавишу `<Enter>`.

В результате будет создан файл `template/index.html` (рис. 6.5), содержащий по умолчанию программный код, приведенный в листинге 6.2.

#### Листинг 6.2. Фрагмент файла `index.html`

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

File Edit View Navigate Code Refactor Run Tools VCS Window Help

Web\_1 > hello > hello > settings.py

```

Project Web_1 C:\Users\Vit\PycharmP...
  - hello
    - firstapp
    - hello
      - __init__.py
      - asgi.py
      - settings.py
      - urls.py
      - wsgi.py
      - templates
        - db.sqlite3
        - manage.py
  - venv library root
External Libraries
Scratches and Consoles

```

**Добавлено** → templates

settings.py

```

51 ]
52
53 ROOT_URLCONF = 'hello.urls'
54
55 TEMPLATE_DIR = os.path.join(BASE_DIR, "templates") Добавлено
56
57 TEMPLATES = [
58   {
59     'BACKEND': 'django.template.backends.django.DjangoTemplates',
60     'DIRS': [TEMPLATE_DIR, ],
61     'APP_DIRS': True,
62     'OPTIONS': {
63       'context_processors': [
64         'django.template.context_processors.debug',
65         'django.template.context_processors.request',
66         'django.contrib.auth.context_processors.auth',
67         'django.contrib.messages.context_processors.messages',
68       ],
69     },
70   },
71 ]

```

Рис. 6.2. Настройка параметров папки templates в файле settings.py

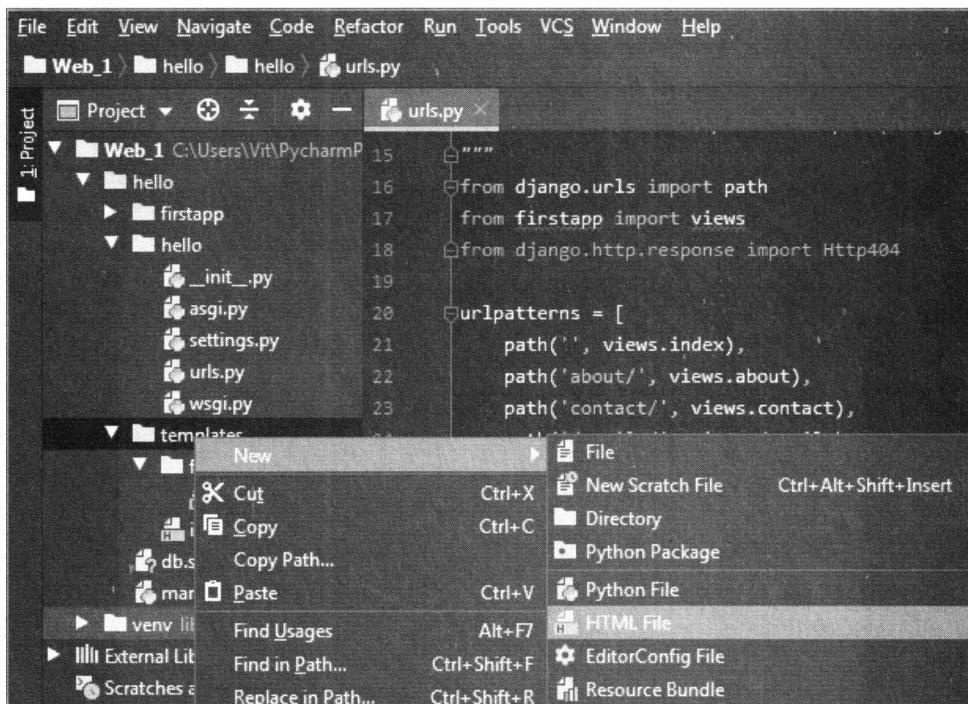


Рис. 6.3. Создание нового HTML-файла в PyCharm

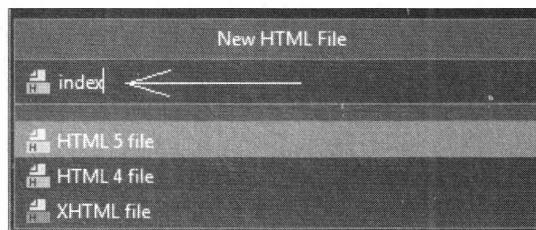


Рис. 6.4. Задание имени нового HTML-файла в PyCharm

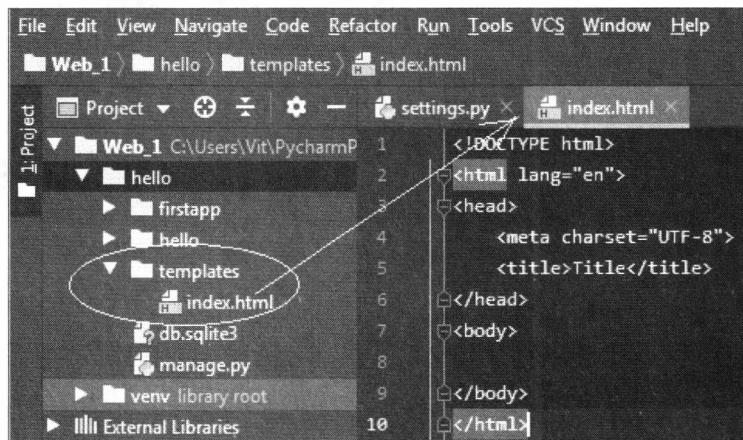


Рис. 6.5. Содержание файла index.html, созданного в PyCharm

Внесем в этот программный код изменения, приведенные в листинге 6.3 (выделены серым фоном).

#### Листинг 6.3. Фрагмент файла index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django!</title>
</head>
<body>
    <h1>Вы на главной странице Django!</h1>
    <h2>templates/index.html</h2>
</body>
</html>
```

После этих изменений наш первый шаблон в окне IDE PyCharm будет выглядеть так, как представлено на рис. 6.6.

По сути, это обычная страница или файл, содержащий код HTML. Теперь используем эту страницу для отправки ответа пользователю. Для этого перейдем в приложении firstapp к файлу firstapp/views.py, который определяет функции для обработки запроса пользователя, и изменим этот файл так, как показано в листинге 6.4.

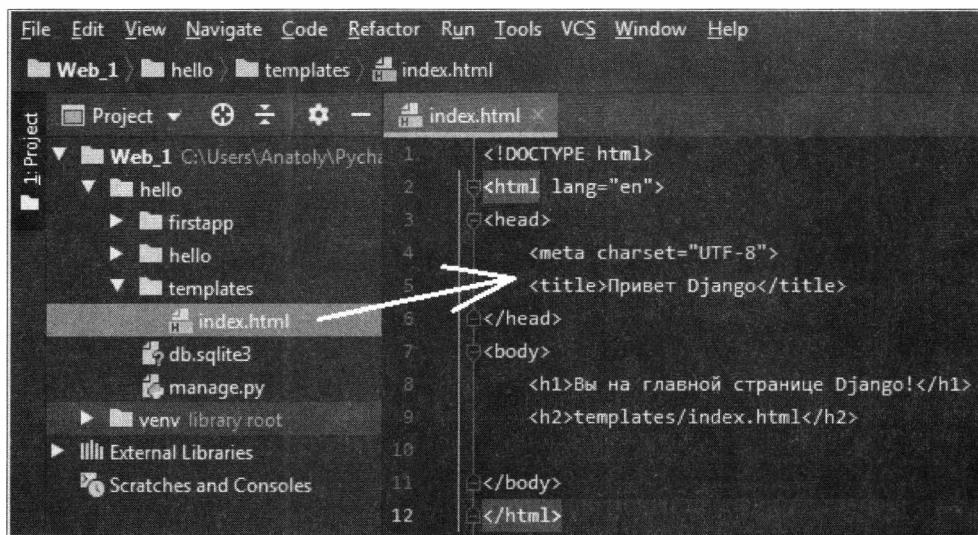


Рис. 6.6. Шаблон index.html в окне IDE PyCharm

**Листинг 6.4. Фрагмент файла views.py**

```

from django.shortcuts import render
def index(request):
    return render(request, "index.html")

```

Что мы здесь сделали? Первым шагом из модуля `django.shortcuts` была импортирована функция `render` (отдавать, предоставлять). Вторым шагом была изменена функция `def index(request)`. Эта функция получает запрос от пользователя (`request`) и в инструкции `return` возвращает ему ответ путем вызова другой функции `render` (рендеринг — отдать, предоставить). В функцию `render` передано два аргумента: запрос пользователя `request` и файл шаблона `index.html`, который находится в папке `templates`. Эти изменения, сделанные в PyCharm, представлены на рис. 6.7.

Проверим, нужно ли вносить изменения в файл `firstrapp/urls.py`, где должно быть прописано сопоставление функции `index` с запросом пользователя к «корню» веб-приложения. Этот файл должен содержать код, представленный в листинге 6.5.

**Листинг 6.5. Фрагмент файла urls.py**

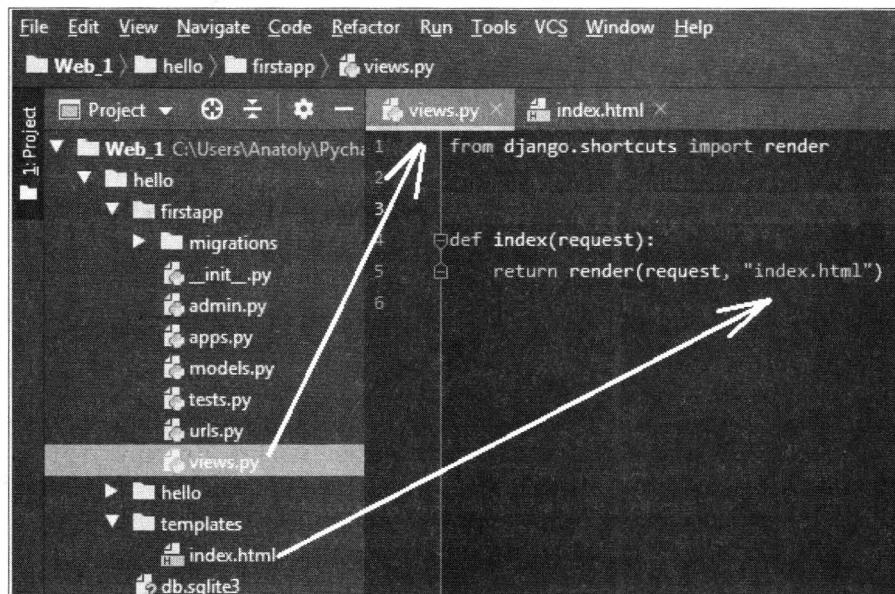
```

from django.urls import path
from firstrapp import views
urlpatterns = [
    path('', views.index),
]

```

Чтобы это проверить, откроем файл `firstrapp/urls.py` (рис. 6.8) и убедимся, что в нем для обращения пользователя к «корню» веб-приложения прописан следующий маршрут:

```
path('', views.index)
```



```

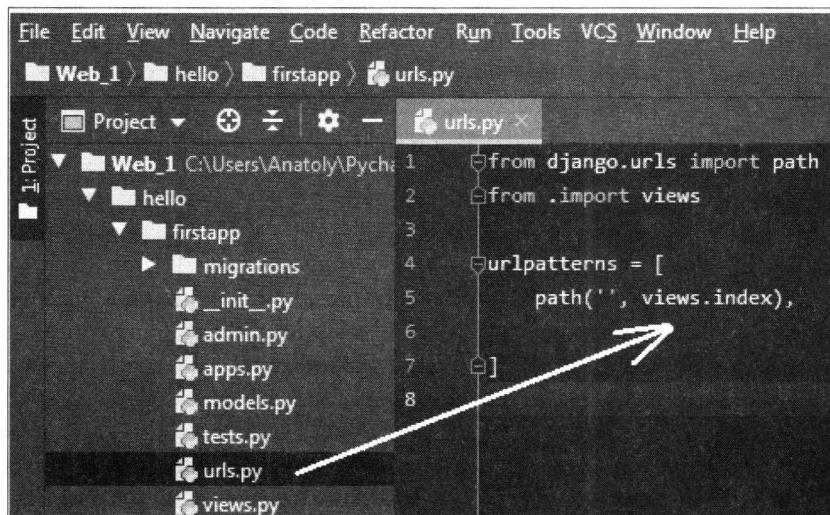
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 > hello > firstapp > views.py
Project views.py index.html
Web_1 C:\Users\Anatoly\PycharmProjects\Web_1
hello
  firstapp
    migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py
    views.py
  templates
    index.html
db.sqlite3

from django.shortcuts import render

def index(request):
    return render(request, "index.html")

```

Рис. 6.7. Изменение содержания файла views.py в программной оболочке PyCharm



```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 > hello > firstapp > urls.py
Project urls.py
Web_1 C:\Users\Anatoly\PycharmProjects\Web_1
hello
  firstapp
    migrations
      __init__.py
      admin.py
      apps.py
      models.py
      tests.py
      urls.py
    views.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index),
]

```

Рис. 6.8. Маршрут перехода к функции index() при обращении пользователя к «корню» веб-приложения

Если маршрут задан верно, то запустим локальный сервер разработки командой  
`python manage.py runserver`

и перейдем в браузер по адресу `http://127.0.0.1:8000/`. Браузер нам отобразит главную страницу сайта (рис. 6.9).

Нас можно поздравить, мы создали свой первый шаблон — это HTML-страница, которая находится в папке `hello/templates/index.html`.

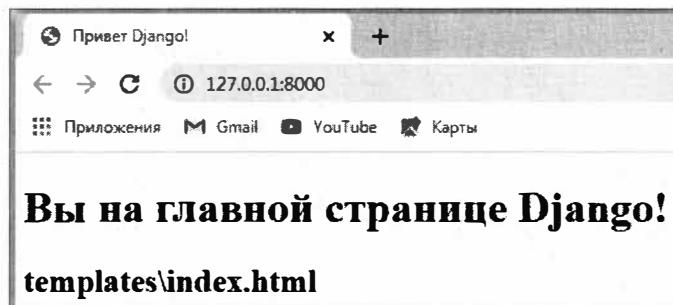


Рис. 6.9. Главная страница сайта, загруженная из шаблона templates\index.html

Пока наш шаблон пустой, т. е. это обычная статичная HTML-страница, но это не беда. Теперь мы можем вносить в этот шаблон изменения, менять его структуру и дизайн, вставлять в него данные, полученные из БД, и возвращать пользователю не статичные, а динамически генерируемые веб-страницы.

Шаблоны Django представляют собой традиционные HTML-файлы, которые дополнены специальными тегами и программным кодом, позволяющим использовать переменные, циклы и другие управляющие структуры. Когда представление вызывает функцию `render()`, она передает данные в шаблон, а шаблон генерирует HTML-код для отображения пользователю. В HTML-страницах, которые являются шаблонами, применяется встроенный язык, который называется языком шаблонов Django или DTL (Django Template Language). С языком шаблонов DTL мы детально познакомимся чуть позже в следующих разделах, а пока разберемся с тем, где и как можно размещать шаблоны.

## 6.2. Создание каталога для шаблонов приложений

Как правило, шаблоны в проекте помещаются в общую папку. Например, в нашем проекте `hello` для размещения шаблонов была создана общая папка с именем `templates`. Однако в проекте Django можно создать несколько приложений, и каждое из них может иметь свой набор шаблонов. Чтобы шаблоны приложений не смешивались, их можно разнести по разным каталогам. В нашем проекте в файле конфигурации `settings.py` в переменной `TEMPLATES` параметр `APP_DIRS` имеет значение `True`. Это говорит о том, что поиск шаблонов будет проводиться не только в папке, указанной в параметре `DIRS` (в нашем случае это папка `templates`), но и в дочерних (вложенных) папках.

В нашем проекте `hello` создано одно приложение — `firstapp`, значит, для этого приложения можно создать собственный каталог с шаблонами. Создадим каталог для шаблонов приложения `firstapp`. Для этого в папке `templates` создадим дочернюю папку `firstapp` (по имени приложения), и в этой папке создадим новый файл — шаблон с именем `home.html`. Место файла `home.html` в структуре каталогов проекта `hello` показано на рис. 6.10.

Внесем в содержание файла `home.html` программный код из листинга 6.6, и он будет выглядеть так, как показано на рис. 6.11.

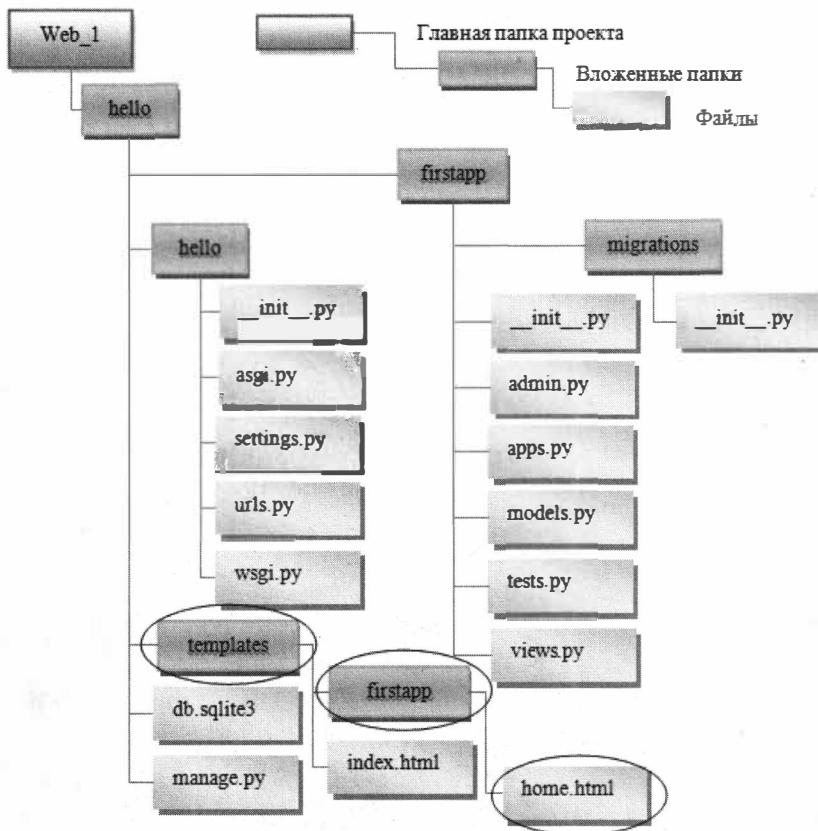


Рис. 6.10. Место файла home.html в структуре каталогов проекта hello

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 > hello > templates > firstapp > home.html
Project  Proj.  + -  home.html
Web_1 C:\Users\Vit\PycharmProjects\HelloWorld\hello\firstapp\templates\firstapp\home.html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Привет Django!</title>
6  </head>
7  <body>
8      <h1>Домашняя страница Django!</h1>
9      <h2>templates/firstapp/home.html</h2>
10     </body>
11 </html>
12

```

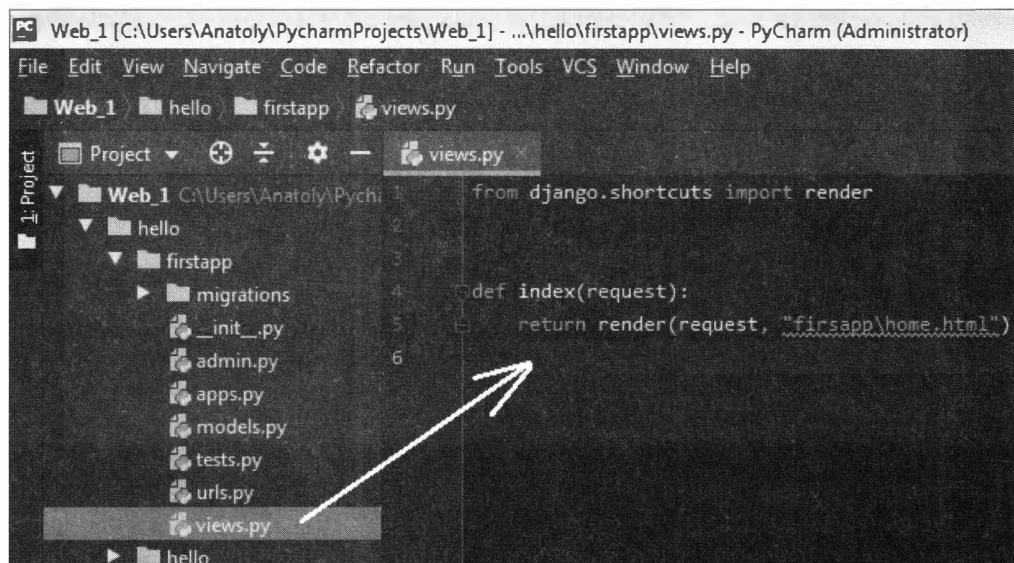
The screenshot shows the 'home.html' file in a code editor. The file content is a simple HTML document. Annotations with arrows point from specific parts of the code to the corresponding files in the project structure. One arrow points from the line '<h2>templates/firstapp/home.html</h2>' to the 'home.html' file located in the 'firstapp/templates/firstapp' directory. Another arrow points from the line 'templates/firstapp/home.html' in the code to the 'home.html' file in the project structure.

Рис. 6.11. Изменения, выполненные в файле home.html

**Листинг 6.6. Фрагмент файла home.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django!</title>
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
</body>
</html>
```

Теперь изменим файл `firstapp/views.py` нашего приложения, указав в нем новый путь к странице, которую нужно выдать пользователю при его обращении к домашней странице сайта — `home.html` (рис. 6.12).



**Рис. 6.12.** Задание пути к странице `home.html`, которую нужно выдать пользователю при его обращении к домашней странице сайта

Теперь функция `index()` будет выглядеть как в листинге 6.7.

**Листинг 6.7. Функция index()**

```
def index(request):
    return render(request, "firstapp\home.html")
```

После этих изменений запустим локальный сервер разработки командой

```
python manage.py runserver
```

и перейдем в браузере по адресу `http://127.0.0.1:8000/` — браузер нам отобразит новую домашнюю страницу сайта `home.html` (рис. 6.13).

Как видно из данного рисунка, шаблон `home.html` был найден и загружен по следующему пути: `templates/firstapp/home.html`.

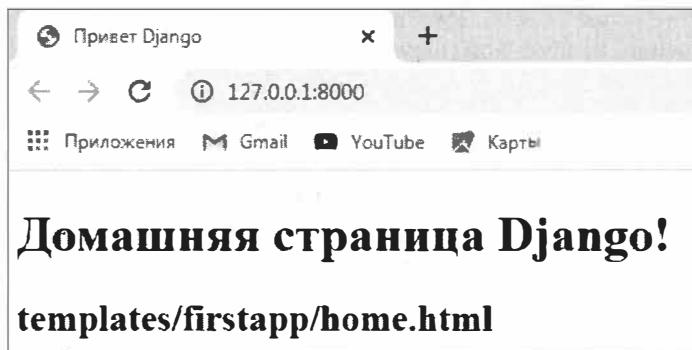


Рис. 6.13. Домашняя страница сайта `home.html`

Теперь в папке `templates/firstapp/` можно размещать шаблоны, которые будут относиться к приложению `firstapp`.

## 6.3. Класс `TemplateResponse`

В предыдущем разделе для загрузки (вызыва) шаблона применялась функция `render` (выводить, отображать), что является наиболее распространенным вариантом. Однако в Django для вызова шаблонов предусмотрен и специальный класс `TemplateResponse` (ответить в шаблоне). Функция `def index(request)` при использовании класса `TemplateResponse` будет выглядеть как в листинге 6.8.

### Листинг 6.8. Функция `index()`

```
from django.template.response import TemplateResponse
def index(request):
    return TemplateResponse(request, "firstapp/home.html")
```

Результат работы приложения с использованием класса `TemplateResponse` будет таким же, как и при вызове функции `render()`.

Итак, мы научились создавать шаблоны HTML-страниц и возвращать их в виде ответов пользователю на их запросы. Но в приводимых примерах мы, по сути, возвращали пустые страницы, что конечному пользователю не совсем интересно. Он ожидает получить в ответ какую-то полезную информацию. Иначе говоря, веб-приложение должно уметь заполнить шаблон страницы теми данными, которые запросил пользователь. Решению этой задачи и будет посвящен следующий раздел.

## 6.4. Язык шаблонов (DTL)

В HTML-страницах, которые являются шаблонами, используется встроенный язык, называемый языком шаблонов Django или DTL (Django Template Language). Синтаксис языка шаблонов включает в себя следующие четыре конструкции:

1. Переменные — {{ variable }}. Имя переменной пишется внутри двойных фигурных скобок. Значение переменной передается в шаблон и отображается на HTML-странице.
2. Теги шаблона — {% tag %}. Теги шаблона заключаются в фигурные скобки со знаками процента и позволяют создавать циклы (инструкция `for`), разветвления (инструкция `if else`), структурные элементы, а также некоторую управляющую логику.
3. Фильтры — {{ variable|filter }}. В шаблонах Django после переменной может быть вставлена вертикальная черта, после которой задается фильтр шаблона. Фильтры шаблонов принимают строку в качестве входных данных и возвращают строку в качестве выходных данных.
4. Комментарии. Однострочный комментарий имеет следующую конструкцию:

```
{ # строка с комментарием #}
```

Для многострочного комментария предназначен специальный тег:

```
{% comment %}
```

**Переменные.** Переменная получает и выводит значение из параметра функции `render`, который называется *контекст* (`context`). Этот параметр представляет собой объект типа словарь, т. е. имеет два атрибута: ключ и значение ключа. Например, строка для вывода имени и фамилии в шаблоне будет выглядеть следующим образом:

Мое имя – {{ first\_name }}. Моя фамилия – {{ last\_name }}.

В параметре `context` функции `render` значения переменным `first_name` и `last_name` можно передать с использованием следующего словаря:

```
{'first_name': 'Михаил', 'last_name': 'Иванов'}
```

В результате пользователь на странице HTML, которая вернется с сервера, получит следующий ответ:

Мое имя – Михаил. Моя фамилия – Иванов.

Если в шаблон передается список, кортеж или словарь, то доступ к соответствующему элементу этих объектов осуществляется по индексу или ключу, которые указываются через точку:

- {{ my\_list.0 }} — для списка;
- {{ my\_tuple.0 }} — для кортежа;
- {{ my\_object.attribute }} — для объекта;
- {{ my\_dict.key }} — для словаря.

**Теги.** Теги обеспечивают произвольную логику в процессе рендеринга. Это определение намерено расплывчато, поскольку с помощью тегов можно реализовать много разнообразных функций. Тег может служить управляющей структурой, например опе-

ратором разветвления `if` или циклом `for`, выводить содержимое из базы данных или даже разрешать доступ к другим тегам шаблона.

Теги имеют следующие обрамляющие символы: `{% tag %}`. Для некоторых тегов требуется начальные и конечные теги, например:

```
{% tag %}  
... tag contents ...  
{% endtag %}
```

Django поставляется с двумя десятками встроенных тегов, о которых можно узнать из документации по Django 4.1.4 по следующей ссылке:

<https://django.readthedocs.io/en/stable/ref/templates/builtins.html#ref-templates-builtins-tags>

Как использовать наиболее употребительные теги, мы далее более детально покажем на конкретных примерах.

**Фильтры.** Фильтры позволяют выполнить преобразование значения переменных и аргументов тегов. Синтаксис фильтров — `{{ variable|filter }}`.

Например, в шаблоне возможен следующий фильтр:

```
{{ value|title }}
```

#### ПРИМЕЧАНИЕ

Термин «фильтр» в Django немного отличается от общепринятого понятия «фильтр». На самом деле фильтры не выполняют фильтрацию данных по какому-либо признаку, они преобразуют значение переменной `value` от одного вида к другому.

Здесь `value` — это переменная, а `title` — это встроенный фильтр, который преобразует первые символы каждого слова строки в заглавный регистр.

Например, в параметре `context` функции `render` через переменную `value` передается в шаблон с использованием словаря следующее значение:

```
{'value': 'это пример использования фильтров в шаблонах.'}
```

Если в шаблоне задан фильтр `{{ value|title }}`, то пользователь на странице HTML, которая вернется с сервера, получит следующий ответ:

**Это Пример Использования Фильтров В Шаблонах.**

Django поставляется с двумя десятками встроенных фильтров, о которых можно узнать из документации по Django 4.1.4 по следующей ссылке:

<https://django.readthedocs.io/en/stable/ref/templates/builtins.html#ref-templates-builtins-filters>

Как применять наиболее употребительные фильтры, мы далее более детально покажем на конкретных примерах.

**Комментарии.** В тело шаблона можно вставить комментарии, которые позволяют сделать код более понятным. Для создания комментария принят следующий синтаксис:

```
{# строка с комментарием #}
```

Комментарий можно размещать в любом месте тела шаблона.

Разберем синтаксис языка DTL на примерах и начнем с переменных.

## 6.5. Передача данных в шаблоны через переменные

Для обозначения переменных в шаблонах Django принят специальный синтаксис: две открывающие фигурные скобки, имя переменной, две закрывающие фигурные скобки. В программном коде это выглядит следующим образом:

```
{{ имя_переменной }}
```

Этот синтаксис выходит за рамки стандартного языка HTML, но Django понимает его при работе с файлами шаблонов. Когда механизм обработки шаблонов Django находит переменную, он заменяет ее значением, содержащимся в контексте. В качестве имени переменной может быть любая комбинация буквенно-цифровых символов и нижнего подчеркивания. Но имя переменной не может начинаться с символа подчеркивания и не может быть числом, также недопустимы пробелы или знаки препинания в именах переменных. Преимуществом шаблонов является то, что можно в рамках одной HTML-страницы передать пользователю совершенно разные данные. Обычно информация в шаблон динамически подгружается из базы данных через представления (views). Поскольку мы еще не рассматривали работу Django в БД, то в последующих примерах будем просто передавать данные в шаблон из переменных Python.

Вернемся к нашему проекту `hello`, содержащему приложение `firstapp`. Добавим в папку `templates/firstapp` еще один шаблон страницы — `index_app1.html` (рис. 6.14).

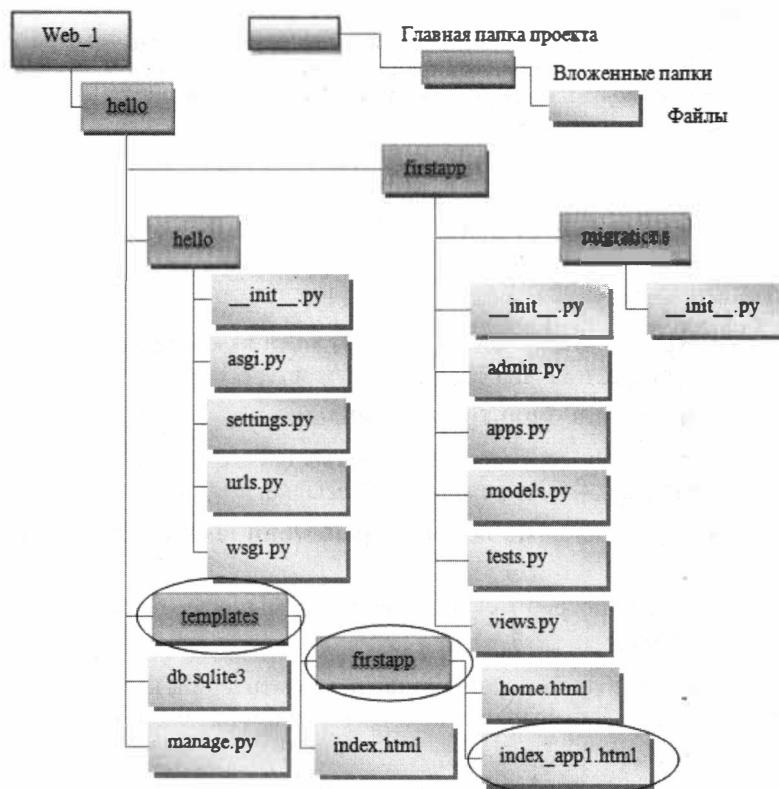


Рис. 6.14. В структуру приложения `firstapp` добавлен шаблон страницы `templates\firstapp\index_app1.html`

Напишем на этой странице программный код, приведенный в листинге 6.9, при этом в окне IDE PyCharm наш новый шаблон будет выглядеть так, как показано на рис. 6.15.

#### Листинг 6.9. Фрагмент файла index\_app1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django!</title>
</head>
<body>
    <h1>{{header}}</h1>
    <p>{{message}}</p>
</body>
</html>
```

Как можно видеть из листинга 6.9, в теге <body> HTML-страницы имеются две переменные, написанные на языке DTL: {{header}} и {{message}}. Эти переменные и будут получать значения из представления (view).

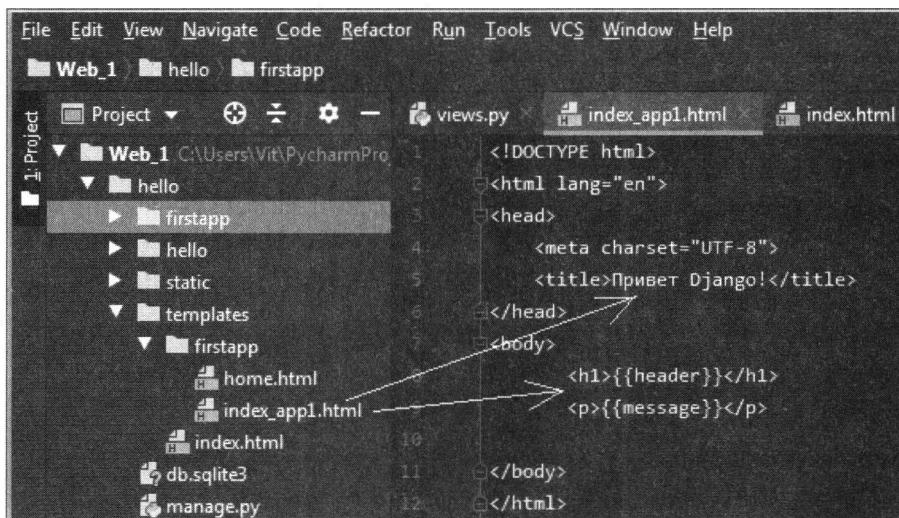


Рис. 6.15. Изменение кода страницы сайта templates\firstapp\index\_app1.html

Чтобы из функции-представления передать данные в шаблон, применяется еще один (третий) параметр в функции render, который называется *контекст* (context). В качестве примера изменим в файле firstapp/views.py функцию def index(), как показано в листинге 6.10. Эти изменения иллюстрирует рис. 6.16.

#### Листинг 6.10. Фрагмент файла views.py

```
def index(request):
    # return render(request, "firstapp/home.html")
```

```

data = {"header": "Передача параметров в шаблон Django",
        "message":
            "Загружен шаблон templates/firstapp/index_app1.html"}
return render(request, "firstapp/index_app1.html", context=data)

```

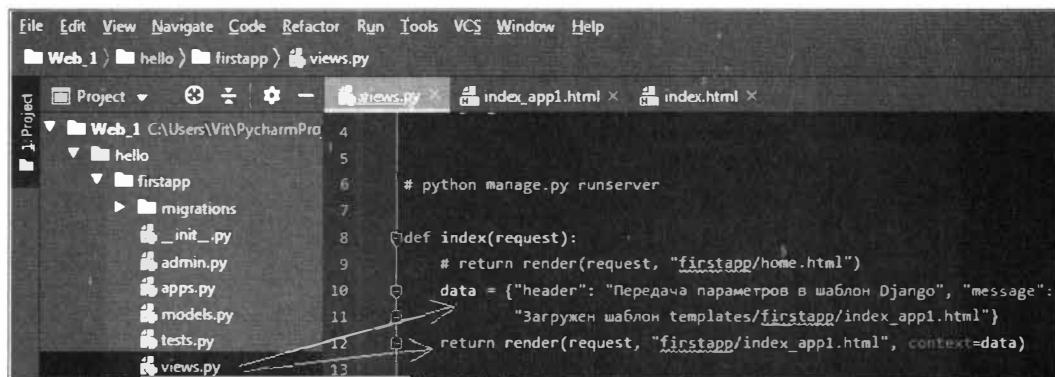


Рис. 6.16. Изменение кода страницы сайта в файле view.py (согласно листингу 6.10)

В нашем шаблоне мы указали две переменные: {{header}} и {{message}} соответственно, в файле firstapp/views.py был создан словарь data, который передается в функцию render через параметр context. Этот словарь имеет два ключа: header и message, а этим ключам мы присвоили следующие значения:

```
{"header": "Передача параметров в шаблон Django",
"message": "Загружен шаблон templates/firstapp/index_app1.html"}
```

После сделанных изменений запустим локальный сервер разработки командой

```
python manage.py runserver
```

и перейдем в браузере по адресу <http://127.0.0.1:8000/>. В результате при обращении к «корню» веб-приложения мы увидим в браузере следующую страницу (рис. 6.17).

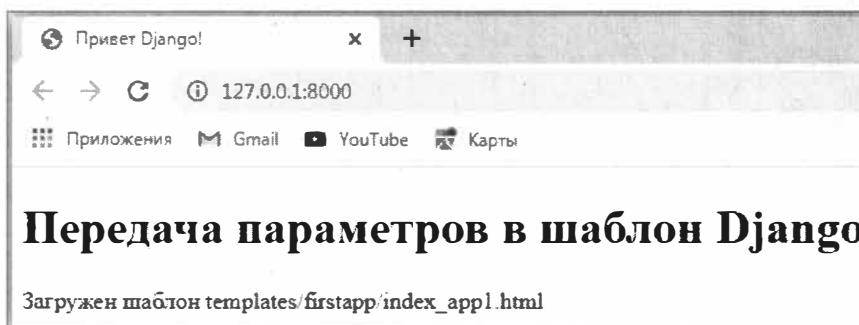


Рис. 6.17. Домашняя страница сайта с данными, подгруженными в шаблон index\_app1.html

Как видно из данного рисунка, в шаблон index\_app1.html вместо переменных {{header}} и {{message}} пользователю были возвращены и показаны значения ключей header и message словаря data.

## 6.6. Передача в шаблон сложных данных

В предыдущем разделе мы встроили в шаблон простые текстовые данные из словаря. Рассмотрим теперь передачу пользователю через шаблон более сложных данных из объектов разного типа: символьная переменная, список, словарь, кортеж. Для этого изменим функцию `def index()` в представлении (в файле `firstapp/views.py`), как показано в листинге 6.11. Эти изменения иллюстрирует рис. 6.18.

### Листинг 6.11. Фрагмент файла views.py

```
def index(request):
    header = "Персональные данные"                      # символьная переменная
    langs = ["Английский", "Немецкий", "Испанский"]      # список
    user = {"name": "Максим,", "age": 30}                  # словарь
    addr = ("Виноградная", 23, 45)                        # кортеж
    data = {"header": header, "langs": langs, "user": user, "address": addr}
    return render(request, "index.html", context=data)
```

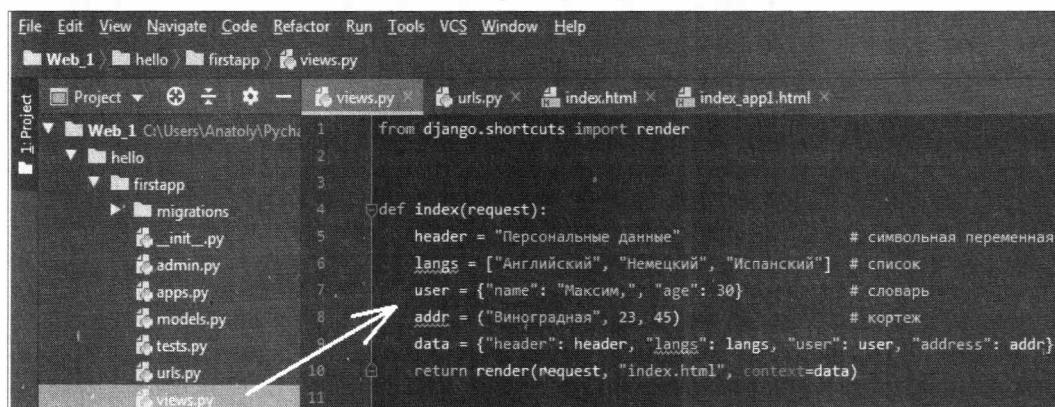


Рис. 6.18. Изменение кода страницы сайта в файле view.py (согласно листингу 6.11)

Здесь мы создали несколько объектов Python: `header` — символьная переменная, `langs` — список, `user` — словарь и `addr` — кортеж. Далее все эти объекты обернули в словарь `data`. Затем в функции `render()` передали словарь `data` третьему параметру — `context`. Обратите внимание, что в функции `render()` используется файл шаблона `index.html`, который находится в корневой папке `templates`.

Теперь нам нужно изменить сам шаблон `templates\index.html`, чтобы он смог принять новые данные (листинг 6.12). Эти изменения показаны на рис. 6.19.

### Листинг 6.12. Фрагмент файла index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```

<title>Передача сложных данных</title>
</head>
<body>
    <h1>{{header}}</h1>
    <p>Имя: {{user.name}} Age: {{user.age}}</p>
    <p>Адрес: ул. {{address.0}}, д. {{address.1}}, кв. {{address.2}}</p>
    <p>Владеет языками: {{langs.0}}, {{langs.1}}</p>
</body>
</html>

```

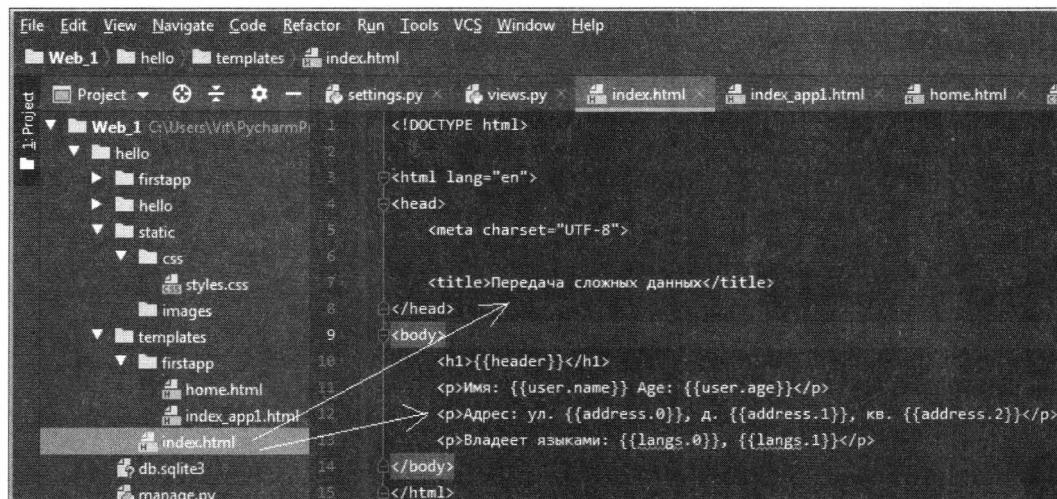


Рис. 6.19. Изменение кода страницы сайта templates/index.html

Поскольку объекты `langs` и `address` представляют соответственно список и кортеж, то мы можем обратиться к их элементам через индексы, как это делается в любом программном коде на Python. К элементам списка, содержащего языки (`langs`), можно обращаться по номеру элемента в списке: `langs.0`, `langs.1`. Соответственно, к элементу кортежа адреса (`address`) мы можем обращаться по индексу элемента в кортеже: `address.0`, `address.1`, `address.2`.

Поскольку объект с информацией о пользователе (`user`) представляет словарь, то мы можем обратиться к его элементам по ключам словаря `name` и `age` следующим образом: `user.name`, `user.age`.

После этих изменений запустим локальный сервер разработки командой

```
python manage.py runserver
```

и перейдем в браузере по адресу <http://127.0.0.1:8000/>. В результате внесенных изменений при обращении к «корню» веб-приложения мы увидим в браузере следующую страницу (рис. 6.20).

Если в файле `firstapp/views.py` в функции `def index()` будет применяться класс `TemplateResponse`, то в его конструктор также через третий параметр можно передать данные для шаблона. В этом случае программный код для этой функции будет выглядеть так, как в листинге 6.13.

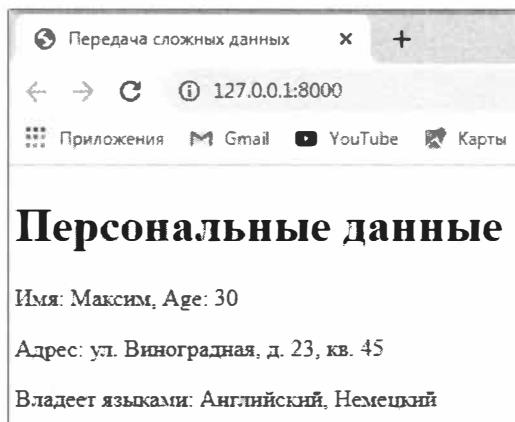


Рис. 6.20. Домашняя страница сайта с данными, подгруженными в шаблон index.html

#### Листинг 6.13. Фрагмент файла views.py

```
from django.template.response import TemplateResponse
def index(request):
    header = "Персональные данные" # символьная переменная
    langs = ["Английский", "Немецкий", "Испанский"] # список
    user = {"name": "Максим,", "age": 30} # словарь
    addr = ("Виноградная", 23, 45) # кортеж
    data = {"header": header, "langs": langs, "user": user, "address": addr}
    return TemplateResponse(request, "index.html", data)
```

Результаты работы этого программного кода будут такими же, как на рис. 6.19.

## 6.7. Использование тегов в шаблонах Django

Рассмотрим несколько примеров наиболее употребительных тегов, в частности тегов для организации циклов (`for`), разветвлений (`if`) и фильтров.

**Тег для организации циклов.** Синтаксис тега для вывода данных в шаблоне в цикле имеет вид:

```
{% for i in list %}
    ...
{% endfor %}
```

Здесь `list` — это список, который передан в шаблон.

В качестве шаблона возьмем файл `templates/firstapp/index_app1.html` и напишем в нем программный код из листинга 6.14.

#### Листинг 6.14. Фрагмент файла index\_app1.html

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
    <meta charset="UTF-8">
    <title>Шаблоны</title>
</head>
<body>
    <h1>{{header}}</h1>
    {% for lang in list_langs %}
        <li>{{ lang }}</li>
    {% endfor %}
</body>
</html>

```

Здесь в теле `<body>` HTML-страницы сформирован шаблон, который имеет два блока. Первый блок содержит переменную `{{header}}`. Во втором блоке организован цикл:

```

{% for lang in list_langs %}
    <li>{{ lang }}</li>
{% endfor %}

```

В первой строке был создан заголовок цикла. В этом заголовке переменная `lang` будет в цикле получать значения иностранных языков, которые содержатся в списке `list_langs`. Вторая строка представляет собой тело цикла. Здесь в теге HTML `<li> ...</li>` (список) содержится шаблон Django с переменной `{{ lang }}`, которая будет последовательно выводить те иностранные языки, которые содержатся в списке `list_langs`. Наконец в третьей строке обозначено завершение тела цикла. Как выглядит программный код этого шаблона в окне PyCharm, показано на рис. 6.21.

Теперь в данный шаблон нужно передать список иностранных языков. Для этого откроем файл представления `firstapp/views.py` и напишем в нем код из листинга 6.15.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 hello templates firstapp index_app1.html
Project Web_1 C:\Users\Anatoly\PycharmProjects\Web_1
hello
firstapp
hello
templates
firstapp
home.html
index_app1.html
index.html
db.sqlite3
manage.py
venv library root
External Libraries

```

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Шаблоны</title>
</head>
<body>
    <h1>{{header}}</h1>
    <% for lang in list_langs %>
        <li>{{ lang }}</li>
    <% endfor %>
</body>
</html>

```

Рис. 6.21. Страница с шаблоном организации цикла в окне IDE PyCharm

**Листинг 6.15. Фрагмент файла views.py**

```
from django.template.response import TemplateResponse
def index(request):
    header = "Иностранные языки" # символьная переменная
    list_langs = ["Английский", "Немецкий", "Испанский",
                  "Французский", "Итальянский"] # список
    data = {"header": header, "list_langs": list_langs}
    return TemplateResponse(request, "firstapp/index_app1.html", data)
```

Здесь создана текстовая переменная `header` и список `list_langs`, который содержит 5 иностранных языков. Эти значения встроены в словарь `data`, который через функцию `request` передан в наш шаблон `firstapp/index_app1.html`.

Как выглядит программный код файла `firstapp/views.py` в окне PyCharm, показано на рис. 6.22.

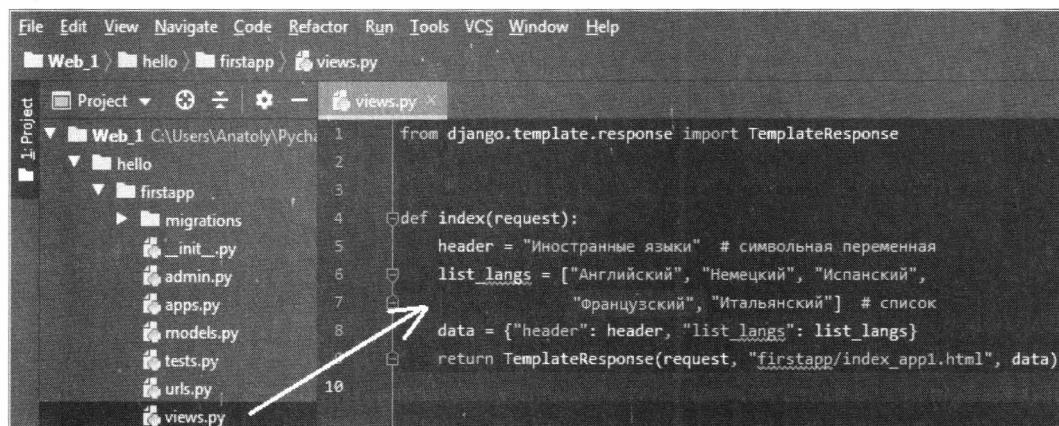


Рис. 6.22. Файл `views.py` с функцией передачи иностранных языков в шаблон Django в окне IDE PyCharm

Итак, у нас все готово для того, чтобы проверить работу тега организации цикла в шаблонах Django. Запустим локальный сервер разработки командой

```
python manage.py runserver
```

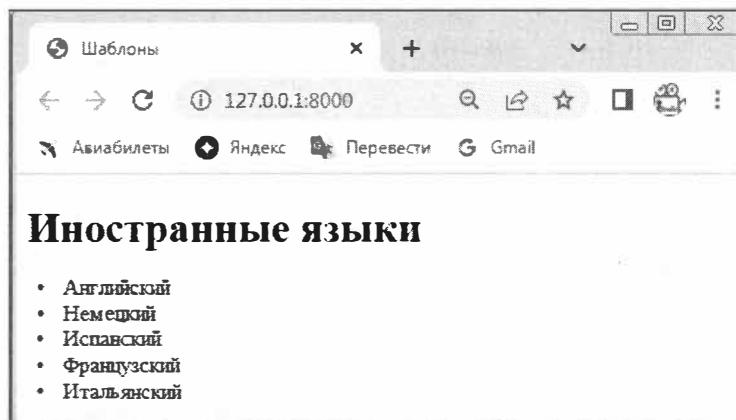
и перейдем в браузере по адресу <http://127.0.0.1:8000/>. В результате выполненных нами действий мы увидим в браузере следующую страницу (рис. 6.23).

Как видно из данного рисунка, добавив тег для организации цикла в шаблон HTML-страницы, мы смогли вывести список иностранных языков.

**Тег для организации разветвлений.** Синтаксис тега для организации разветвлений в шаблоне имеет вид:

```
{% if <логическое_выражение> %}
    {{ переменная1 }}
{% elif <логическое_выражение> %}
    {{ переменная2 }}
```

```
{% else %}
    {{ переменная3 }}
{% endif %}
```



**Рис. 6.23.** Вывод списка иностранных языков с использованием циклов в шаблонах Django

В качестве шаблона выберем тот же файл templates/firstapp/index\_app1.html. Напишем в этом файле программный код, приведенный в листинге 6.16.

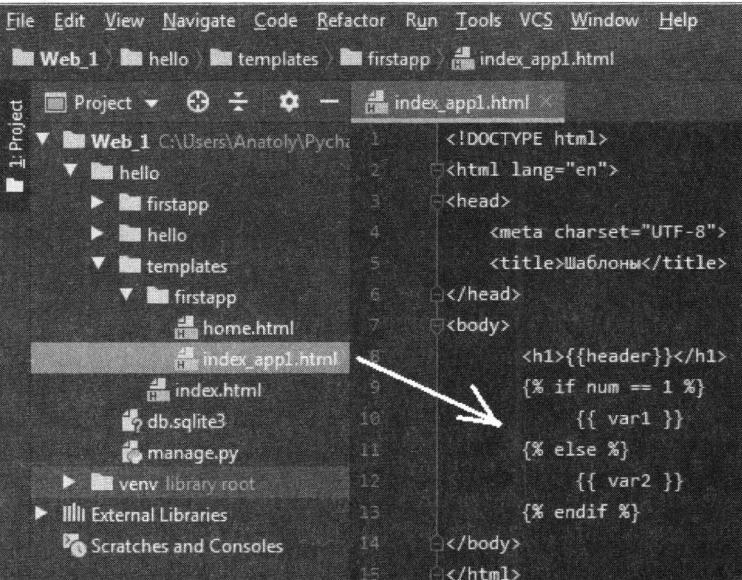
#### Листинг 6.16. Фрагмент файла index\_app1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Шаблоны</title>
</head>
<body>
    <h1>{{header}}</h1>
    {% if num == 1 %}
        {{var1}}
    {% else %}
        {{var2}}
    {% endif %}
</body>
</html>
```

Здесь в теле <body> HTML-страницы сформирован шаблон с двумя блоками. Первый блок содержит переменную {{header}}. Во втором блоке находится инструкция if, которая организует разветвление программы на две ветки:

```
{% if num == 1 %}
    {{var1}}
{% else %}
    {{var2}}
{% endif %}
```

В первой строке был создан заголовок инструкции `if`, в котором находится логическое выражение `num == 1`. Если `num` будет равно 1, то пользователю вернется значение переменной `var1`, а если `num` будет иметь любое другое значение, то пользователю вернется значение переменной `var2`. В последней строке обозначено завершение тела блока `if`. Как выглядит программный код этого шаблона в окне PyCharm, показано на рис. 6.24.



```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Шаблоны</title>
</head>
<body>
<h1>{{header}}</h1>
{% if num == 1 %}
{{ var1 }}
{% else %}
{{ var2 }}
{% endif %}
</body>
</html>
```

Рис. 6.24. Страница с шаблоном организации разветвления в окне IDE PyCharm

Теперь в этот шаблон нужно передать значения переменных `header`, `num`, `var1` и `var2`. Для этого откроем файл представления `firstapp/views.py` и напишем в нем код из листинга 6.17.

#### Листинг 6.17. Фрагмент файла views.py

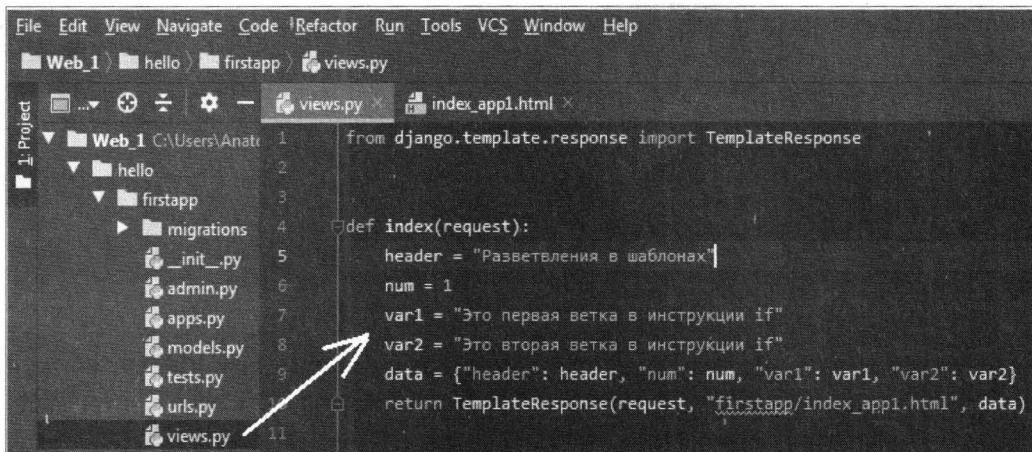
```
from django.template.response import TemplateResponse

def index(request):
    header = "Разветвления в шаблонах"
    num = 1
    var1 = "Это первая ветка в инструкции if"
    var2 = "Это вторая ветка в инструкции if"
    data = {"header": header, "num": num, "var1": var1, "var2": var2}
    return TemplateResponse(request, "firstapp/index_app1.html", data)
```

Здесь созданы три текстовые переменные `header`, `var1`, `var2` и числовая переменная `num`. Эти значения встроены в словарь `data`, который через функцию `request` предан в наш шаблон `firstapp/index_app1.html`. Как выглядит программный код файла `firstapp/views.py` в окне PyCharm, показано на рис. 6.25.

Итак, у нас все готово для того, чтобы проверить работу тега организации разветвления в шаблонах Django. Запустим локальный сервер разработки командой

```
python manage.py runserver
```



```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 hello firstapp views.py index_app1.html
Project Web_1 C:\Users\Anatoly\PycharmProjects\Web_1\hello\firstapp\views.py 1 from django.template.response import TemplateResponse
2 .
3 .
4 def index(request):
5     header = "Разветвления в шаблонах"
6     num = 1
7     var1 = "Это первая ветка в инструкции if"
8     var2 = "Это вторая ветка в инструкции if"
9     data = {"header": header, "num": num, "var1": var1, "var2": var2}
10    return TemplateResponse(request, "firstapp/index_app1.html", data)
11

```

Рис. 6.25. Файл views.py с функцией передачи значений переменных в шаблон Django в окне IDE PyCharm

и перейдем в браузере по адресу <http://127.0.0.1:8000/>. В результате выполненных нами действий мы увидим в браузере следующую страницу (рис. 6.26).

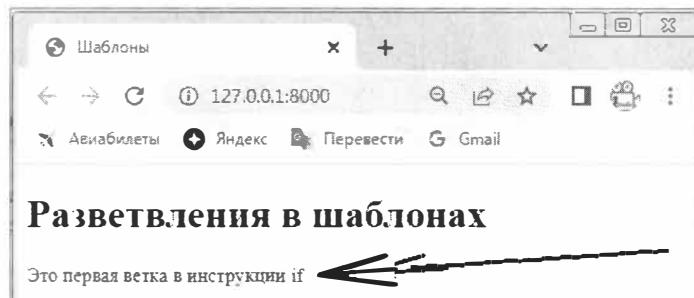


Рис. 6.26. Страница из шаблона Django с разветвлением при num=1

Как видно из данного рисунка, при `num=1` шаблон вернул пользователю значение переменной `var1`, т. е. сработала первая ветка разветвления в инструкции `if`. Если теперь в файле `firstapp/views.py` изменить значение переменной `num`, например `num=2`, то шаблон вернет пользователю значение переменной `var2`, т. е. сработает вторая ветка разветвления в инструкции `if`, и пользователю вернется страница, представленная на рис. 6.27.

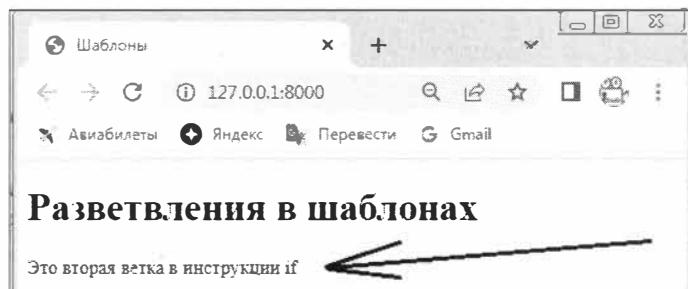


Рис. 6.27. Страница из шаблона Django с разветвлением при num=2

**Тег для создания фильтров.** В шаблонах Django можно использовать фильтры. Обычно фильтры позволяют выделить некоторые элементы из группы, но в шаблонах Django фильтры выполняют иную функцию — они позволяют преобразовать значения переменных и аргументов тегов, т. е. можно, например, изменить регистр символов, задать формат даты, увеличить значение переменной и т. п.

Синтаксис тега для создания фильтров в шаблоне имеет вид:

```
{% variable_name | filter_name %}
```

Здесь:

- `variable_name` — имя переменной, которую нужно преобразовать;
- `filter_name` — имя фильтра.

Из фильтров можно создать «цепочку», при этом выходные данные одного фильтра являются входными данными для следующего, например `{% text|title|length %}`. В Django достаточно много встроенных фильтров, но мы для примера рассмотрим только некоторые из них:

- `add` — добавляет аргумент к значению;
- `date` — преобразует дату в соответствии с заданным форматом;
- `time` — преобразует время в соответствии с заданным форматом;
- `title` — преобразует строку в регистр заголовков, заставляя слова начинаться с символа верхнего регистра, а остальные символы — со строчных букв;
- `upper` — преобразует все символы строки в верхний регистр.

В качестве шаблона возьмем тот же файл `templates/firstapp/index_app1.html` и напишем в нем программный код из листинга 6.18.

#### Листинг 6.18. Фрагмент файла index\_app1.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Шаблоны</title>
</head>
<body>
    <h1>{{header}}</h1>
    <li>{{value_num|add:"2"}}</li>
    <li>{{value_date|date:"D d M Y"}}</li>
    <li>{{value_time|time:"H:i"}}</li>
    <li>{{value_title|title}}</li>
    <li>{{value_upper|upper}}</li>
</body>
</html>
```

Здесь в теле `<body>` HTML-страницы сформирован шаблон из шести строк. Первая строка содержит переменную `{{header}}`. В остальных пяти строках находятся различные фильтры. Первая строка заключена в тег HTML `<h1></h1>` — заголовок, остальные стро-

ки обрамлены тегами HTML `<li></li>` — список. Как выглядит программный код этого шаблона в окне PyCharm, показано на рис. 6.28.

Теперь в этот шаблон нужно передать значения переменных `header`, `value_num`, `value_date`, `value_time`, `value_title`, `value_upper`. Для этого откроем файл представления `firstapp/views.py` и напишем в нем код из листинга 6.19.

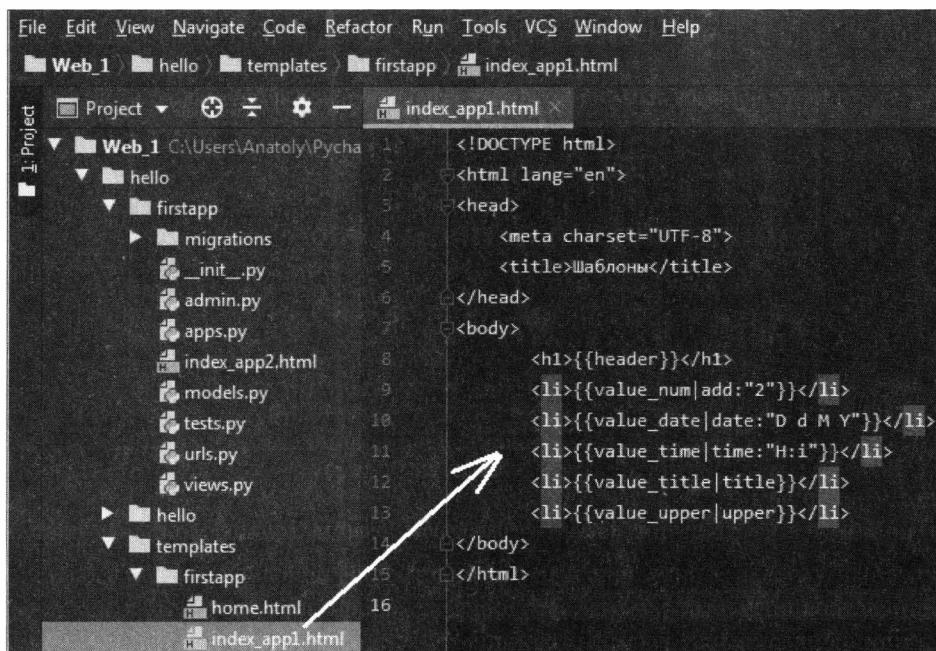


Рис. 6.28. Страница с шаблоном, содержащим фильтры, в окне IDE PyCharm

#### Листинг 6.19. Фрагмент файла views.py

```

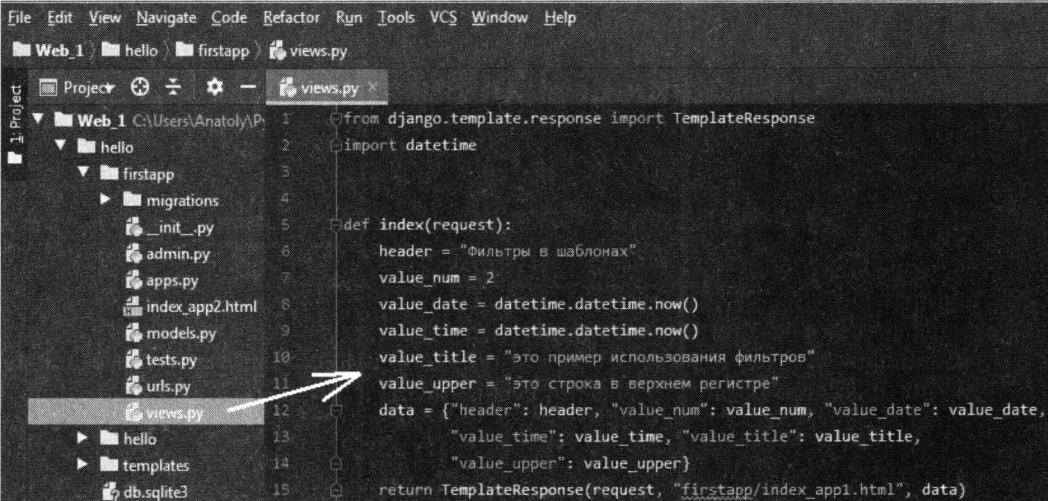
from django.template.response import TemplateResponse
import datetime

def index(request):
    header = "Фильтры в шаблонах"
    value_num = 2
    value_date = datetime.datetime.now()
    value_time = datetime.datetime.now()
    value_title = "это пример использования фильтров"
    value_upper = "это строка в верхнем регистре"
    data = {"header": header, "value_num": value_num, "value_date": value_date,
            "value_time": value_time, "value_title": value_title,
            "value_upper": value_upper}
    return TemplateResponse(request, "firstapp/index_app1.html", data)

```

Здесь был импортирован модуль Python `datetime`, который позволяет получить текущую дату и время. Затем созданы три текстовые переменные `header`, `value_title`, `value_upper`,

числовая переменная `value_num` и две переменные, содержащие текущую дату и время — `value_date`, `value_time`. Эти значения встроены в словарь `data`, который через функцию `request` передан в наш шаблон `firstapp/index_app1.html`. Как выглядит программный код файла `firstapp/views.py` в окне PyCharm, показано на рис. 6.29.



```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
Web_1 hello firstapp views.py
Project Web_1 C:\Users\Anatoly\P...
  -> Web_1
    -> hello
      -> firstapp
        migrations
          __init__.py
          admin.py
          apps.py
          index_app2.html
          models.py
          tests.py
          urls.py
          views.py
        hello
        templates
        db.sqlite3
1   from django.template.response import TemplateResponse
2   import datetime
3
4
5   def index(request):
6       header = "Фильтры в шаблонах"
7       value_num = 2
8       value_date = datetime.datetime.now()
9       value_time = datetime.datetime.now()
10      value_title = "это пример использования фильтров"
11      value_upper = "это строка в верхнем регистре"
12      data = {"header": header, "value_num": value_num, "value_date": value_date,
13              "value_time": value_time, "value_title": value_title,
14              "value_upper": value_upper}
15
16      return TemplateResponse(request, "firstapp/index_app1.html", data)

```

Рис. 6.29. Файл `views.py` с функцией передачи значений переменных в шаблон Django в окне IDE PyCharm

Итак, у нас все готово для того, чтобы проверить работу тега с фильтрами в шаблонах Django. Запустим локальный сервер разработки командой

```
python manage.py runserver
```

и перейдем в браузере по адресу <http://127.0.0.1:8000/>. В результате выполненных действий мы увидим в браузере следующую страницу (рис. 6.30).

Из данного рисунка видно, что с использованием фильтров были получены следующие результаты:

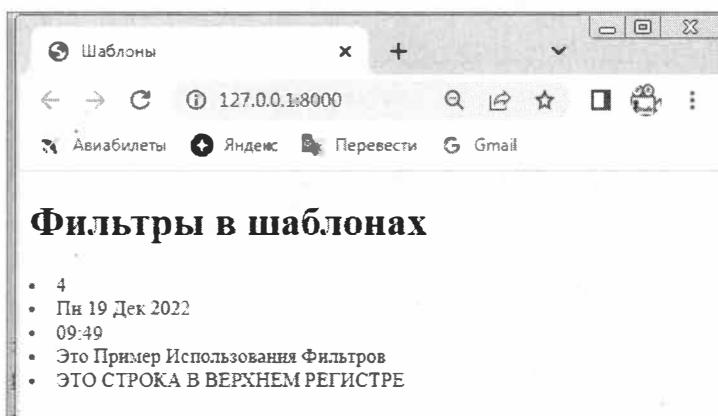


Рис. 6.30. Страница из шаблона Django с результатами работы фильтров

- фильтр `add` к переданному в шаблон числу 2 прибавил 2, и получилось число 4;
- фильтр `date` из переменной, содержащей дату и время, выбрал только дату;
- фильтр `time` из переменной, содержащей дату и время, выбрал только время;
- фильтр `title` перевел все первые символы слов в верхний регистр;
- фильтр `upper` перевел все символы слов в верхний регистр.

## 6.8. Статические файлы в шаблонах Django

Пришло время ближе познакомиться со статическими файлами. *Статическими файлами* называются все файлы каскадных таблиц стилей (Cascading Style Sheets, CSS), изображений, а также скриптов JavaScript, т. е. файлы, содержимое которых не зависит от контекста запроса и одинаково для всех пользователей. Можно воспринимать их как своего рода «украшение» для веб-страниц, а значит, и для придания привлекательного вида шаблонам.

### 6.8.1. Основы каскадных таблиц стилей

Для придания привлекательности информации, выводимой на HTML-страницах, используются различные стили форматирования текстов, оформленные в виде *каскадных таблиц стилей* (CSS) с помощью специального языка. Такой подход обеспечивает возможность прикреплять стиль (тип шрифта, его размер, цвет и пр.) к структурированным документам. Обычно CSS-стили служат для создания и изменения стиля элементов веб-страниц и пользовательских интерфейсов, написанных на языках HTML, но также могут быть применены к любому виду XML-документа. Отделяя стиль представления от содержимого документов, CSS упрощает создание веб-страниц и обслуживание сайтов.

Объявление стиля состоит из двух частей: *селектора* и *объявления*. В HTML имена элементов нечувствительны к регистру, поэтому в селекторе значение `h1` работает так же, как и `H1`. Объявление состоит из двух частей: имени свойства (например, `color`) и значения свойства (например, `grey`). Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) указываются форматирующие команды — свойства и их значения (рис. 6.31).



Рис. 6.31. Задание стилей на HTML-страницах

Стили могут быть следующих типов:

- внутренние — располагаются в заголовке `<head>` HTML-страницы и относятся ко всем ее элементам;

- встроенные — располагаются в теле <body> HTML-страницы и предназначены для стилизации конкретного HTML-элемента, а не всего HTML-кода;
- внешняя таблица стилей — представляет собой внешний CSS-файл, который связан с HTML-страницей.

Все три стиля CSS могут присутствовать на одной веб-странице. Однако нужно учитывать приоритет одного типа стилей над другими. В частности, эти стили имеют следующие приоритеты: встроенный --> внутренний --> внешний.

CSS-код *внутренних стилей* располагается в заголовке <head> HTML-файла и задается с помощью атрибута `style`, например:

```
<style>h1{color: green;}</style>
```

При этом все заголовки текста на HTML-странице будут иметь зеленый цвет.

CSS-код *встроенных стилей* располагается в HTML-файле непосредственно внутри тега элемента с помощью атрибута `style`, например:

```
<p style="color: red;">Это текст красного цвета!</p>
```

При этом текст в данном теге будет иметь красный цвет.

Рассмотрим использование стилей на нескольких примерах. Для этого вернемся к нашему проекту `hello` и перейдем к файлу `hello\templates\firstapp\home.html`, который содержит следующий текст (листинг 6.20).

#### Листинг 6.20. Фрагмент файла `home.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
</body>
</html>
```

Откроем файл `firstapp\views.py` и изменим значение функции `def index()`, как показано в листинге 6.21.

#### Листинг 6.21. Фрагмент файла `views.py`

```
from django.shortcuts import render

def index(request):
    return render(request, "firstapp/home.html")
```

Теперь, если запустить локальный веб-сервер:

```
python manage.py runserver
```

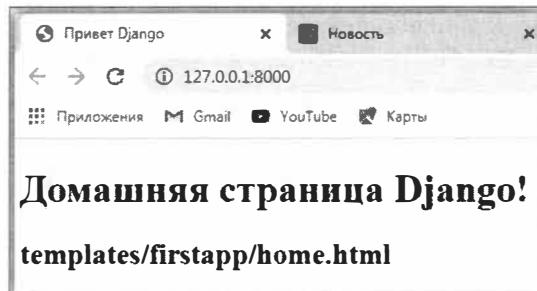


Рис. 6.32. Вывод страницы home.html без использования стилей

и перейти на главную страницу `http://127.0.0.1:8000/`, то мы увидим следующий результат (рис. 6.32).

Изменим стиль выводимого текста: первую строку выведем красным цветом, а вторую — синим, при этом будем использовать внутренний стиль в заголовке HTML-страницы. Для этого изменим текст файла `templates\firstapp\home.html` следующим образом (листинг 6.22).

#### Листинг 6.22. Фрагмент файла home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>h1{color: red;}</style>
    <style>h2{color: blue; font-size: 22px;}</style>
    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
</body>
</html>
```

Здесь мы в стилях для тегов `<h1>` задали красный цвет, а для тегов `<h2>` — синий цвет и размер шрифта 22 px. Снова обратимся к странице `templates\firstapp\home.html` и теперь получим следующий результат (рис. 6.33).

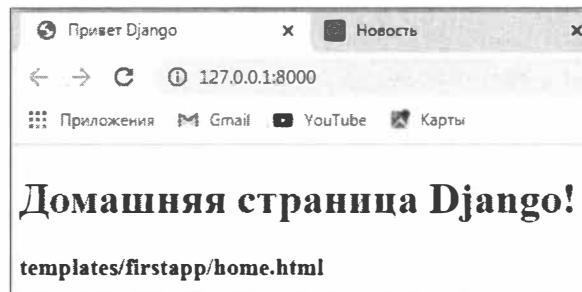


Рис. 6.33. Вывод страницы home.html с использованием внутреннего стиля

Как видно из данного рисунка (при выводе цветного изображения на экран компьютера), цвета выводимого текста поменялись: первая строка выведена красным цветом, а вторая — синим.

*Встроенные стили* отличаются от внутренних стилей тем, что они встраиваются в тело <body> HTML-документа, при этом стили встраиваются в другие теги. Изменим текст файла templates\firstapp\home.html, как показано в листинге 6.23.

#### Листинг 6.23. Фрагмент файла home.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>h1{color: red;}</style>
    <style>h2{color: blue; font-size: 22px;}</style>

    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1 style="color: green">Домашняя страница Django!</h1>
    <h2 style="color: red">templates/firstapp/home.html</h2>
</body>
</html>
```

Здесь мы в теге заголовка <head> для текста с тегами h1 оставили красный цвет, с тегами h2 — синий. Однако в теле HTML-страницы <body> переопределили эти цвета с использованием встроенных стилей. Для заголовка h1 задали зеленый цвет, а для заголовка h2 — красный. Обратите внимание, что стили встроены непосредственно в теги h1 и h2.

Снова обратимся к странице templates\firstapp\home.html и теперь получим следующий результат (рис. 6.34).

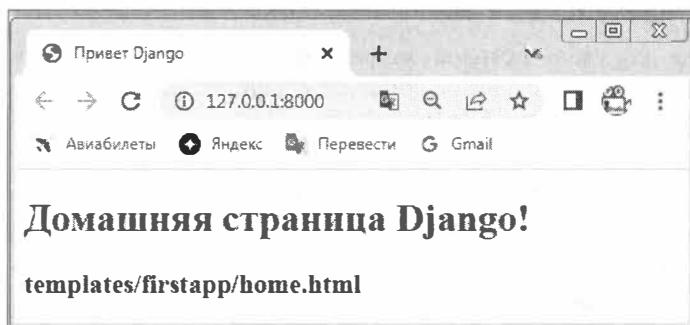


Рис. 6.34. Вывод страницы home.html с использованием встроенного стиля

Как видно из данного рисунка, встроенные стили, которые находятся непосредственно в тегах тела HTML-страницы, переопределили цвета, которые были заданы во внутренних стилях заголовка HTML-страницы. Это подтверждает приоритет встроенных стилей над внутренними.

Внутренние и встроенные стили достаточно просты, но не совсем удобны. Если через какое-то время потребуется изменить цвет всех заголовков на всех страницах, то нужно будет менять код на каждой из страниц. Такой ситуации можно избежать, если для стилизации сформировать внешний CSS-файл, что является наиболее эффективным методом стилизации при разработке больших веб-сайтов. Внешний файл \*.css можно создать с помощью любого текстового редактора, а потом просто связать HTML-страницы с этим файлом.

Итак, *внешняя таблица стилей* представляет собой текстовый файл с расширением `css`. В этом файле находится набор CSS-стилей для различных элементов HTML-страниц. Внутри такого файла содержатся только стили, без HTML-разметки. Внешняя таблица стилей подключается к веб-странице с помощью тега `<link>`, расположенного внутри заголовка HTML-страницы, т. е. в теге `<head>`, например:

```
<head>
<link rel="stylesheet" href="css/style.css">
</head>
```

Такие стили работают для всех страниц сайта. Использованию внешних таблиц стилей и других статичных файлов посвящен следующий раздел.

## 6.8.2. Использование статических файлов в шаблонах Django

В веб-приложениях, как правило, присутствуют различные статические файлы — это изображения, файлы каскадных таблиц стилей (CSS), скрипты JavaScript, тексты и т. п. Рассмотрим, как мы можем использовать подобные файлы в шаблонах Django.

Обратимся к проекту из предыдущего раздела, где рассматривались вопросы создания шаблонов, и добавим в корневую папку проекта новую вложенную папку `static`. А чтобы не смешивать в одной папке различные типы статичных файлов, создадим для каждого типа файлов отдельную папку. В частности, создадим в папке `static` папку `images` — для изображений, и папку `css` — для таблиц стилей. Подобным образом можно создавать папки и для файлов других типов. Теперь структура нашего проекта будет выглядеть так, как показано на рис. 6.35.

Создадим в папке для таблиц стилей `css` файл `styles.css`. Для этого в окне программной оболочки PyCharm щелкните правой кнопкой мыши на папке `css` и из появившегося меню выполните команду `New | File` (рис. 6.36).

После нажатия на клавишу `<Enter>` откроется новое окно, в котором нужно набрать имя создаваемого файла: `styles.css` (рис. 6.37).

Внесите в файл `styles.css` программный код из листинга 6.24, как показано на рис. 6.38.

### Листинг 6.24. Файл `styles.css`

```
body h1 {color: red;}
body h2 {color: green;}
```

Теперь используем этот файл в шаблоне. Для этого в начале файла шаблона необходимо определить инструкцию:

```
{% load static %}
```

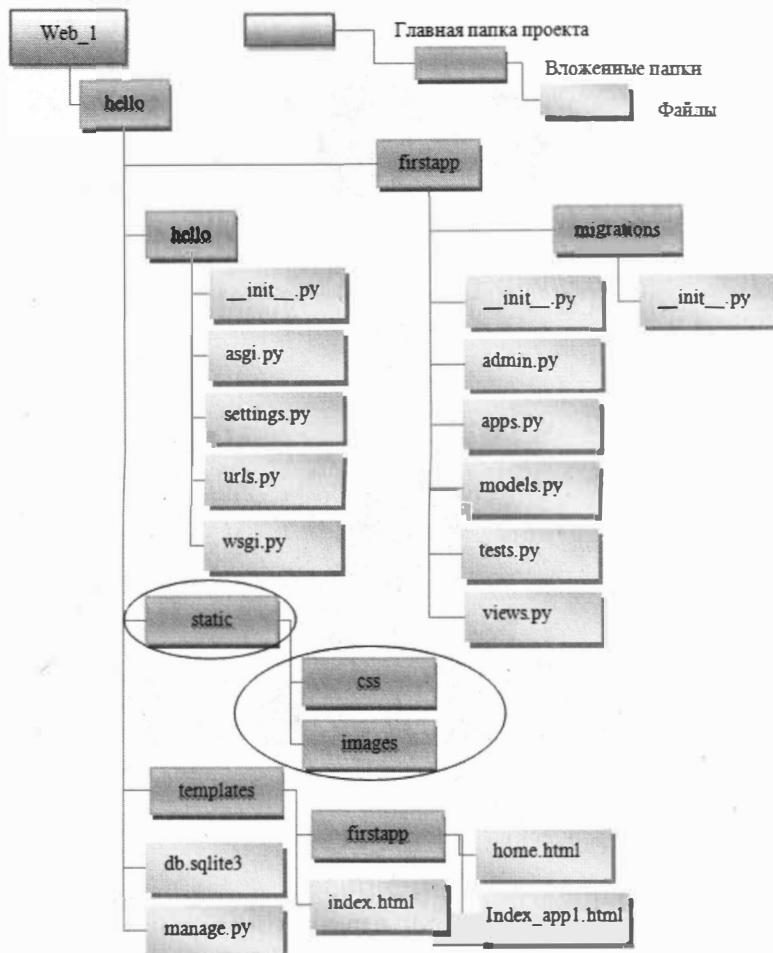


Рис. 6.35. Структура проекта `hello` после добавления в него папок для размещения статичных файлов

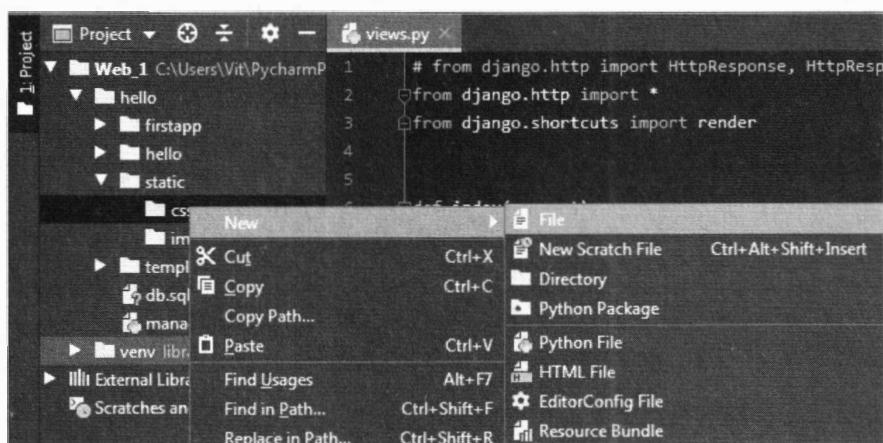


Рис. 6.36. Вход в режим создания нового файла в папке для таблиц стилей css

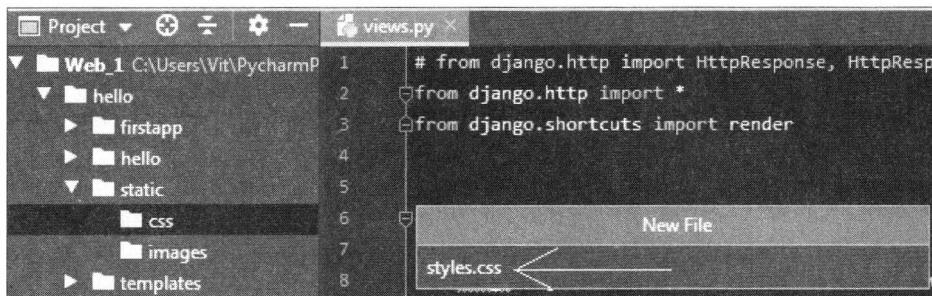


Рис. 6.37. Задание имени новому файлу в папке для таблиц стилей css

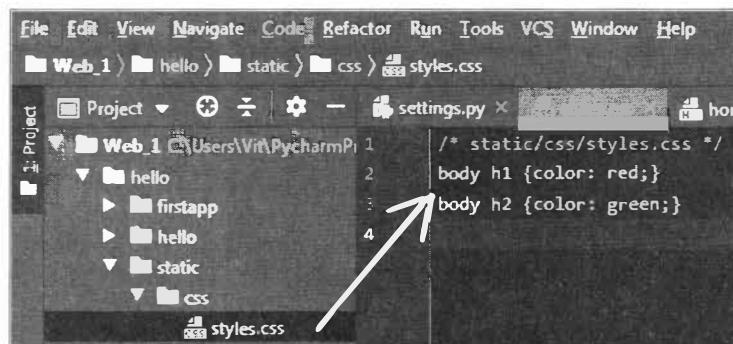


Рис. 6.38. Задание атрибутов в файле таблицы стилей styles.css

Для определения пути к статическим файлам применяются выражения такого типа:

```
{% static "путь к файлу внутри папки static" %}
```

В качестве шаблона снова возьмем HTML-страницу templates\firstapp\home.html. Изменим код на этой странице так, как показано в листинге 6.25 и на рис. 6.39.

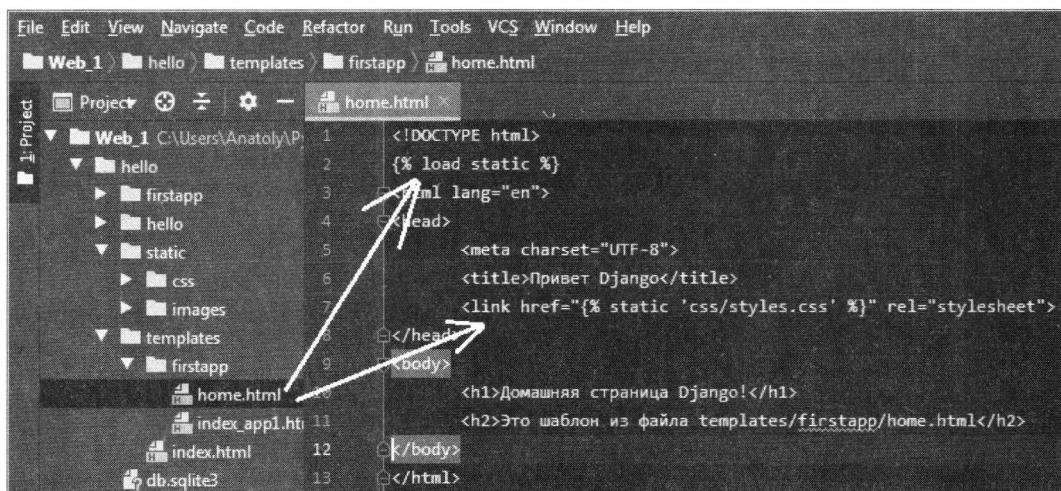


Рис. 6.39. Место инструкций load static и link в файле шаблона home.html

**Листинг 6.25. Фрагмент файла home.html**

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет' Django</title>
    <link href="{% static 'css/styles.css' %}" rel="stylesheet">
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>Это шаблон из файла templates/firstapp/home.html</h2>
</body>
</html>
```

Как видно из данного рисунка, в шаблоне присутствует тег `{% load static %}`, загружающий в шаблон URL-адрес хранилища, в котором находятся статические файлы. Для определения пути к статическим файлам в шаблоне используются выражения типа:

```
{% static "путь к файлу внутри папки static" %}
```

Так как мы будем размещать CSS-файлы в папке `css/styles.css`, то этот тег у нас выглядит следующим образом:

```
{% static 'css/styles.css' %}
```

Тег `<link>` (от англ. link — ссылка, связь) устанавливает связь с внешним документом вроде файла со стилями или изображениями. Атрибут `rel="stylesheet"` указывает, что этот файл является таблицей стилей. Таким образом, для `href` мы фактически задали путь к CSS-файлу.

Чтобы файлы из папки `static` могли использоваться в приложениях, нужно указать путь к этой папке в файле `settings.py`, добавив в конец файла код из листинга 6.26, как показано на рис. 6.40.

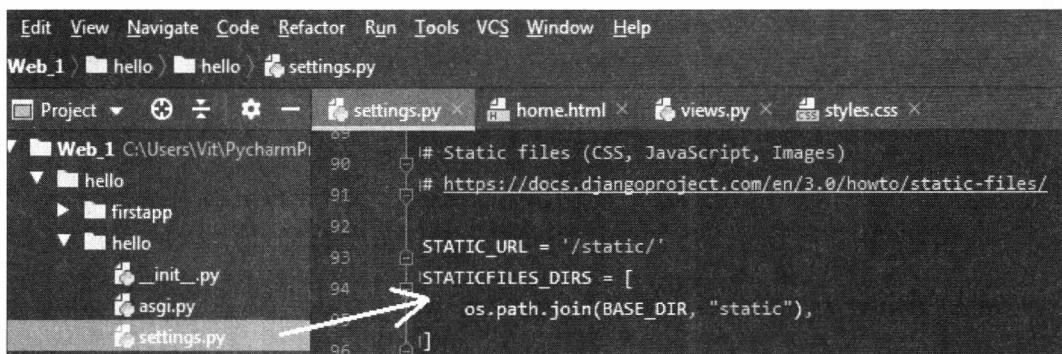


Рис. 6.40. Указание пути к папке static

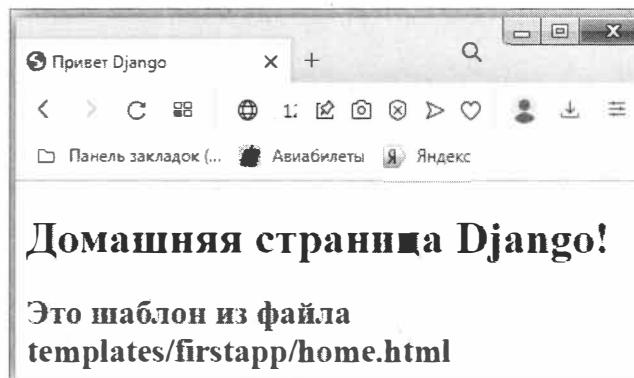
**Листинг 6.26. Фрагмент файла settings.py**

```
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, "static"),
]
```

Если после этих изменений запустим локальный веб-сервер

```
python manage.py runserver
```

и перейдем на главную страницу **http://127.0.0.1:8000/**, то мы увидим следующий результат (рис. 6.41).



**Рис. 6.41.** Отображение текста на странице home.html в соответствии со стилями, взятыми из файла styles.css

**ПРИМЕЧАНИЕ**

В некоторых версиях Google Chrome могут возникнуть проблемы с использованием подключенных CSS-файлов. Если у вас в Google Chrome вместо цветного текста на этой странице появился текст в черно-белом цвете, то попробуйте загрузить ее в браузер Оргея.

Как видно из данного рисунка (при выводе цветного изображения на экран компьютера), в соответствии со стилями, указанными в файле styles.css, текст тега h1 выделен красным цветом, а текст тега h2 — зеленым. При этом в самом HTML-файле стили для выделения текста каким-либо цветом не указаны, а лишь сделаны ссылки на файл со стилями. Такой подход очень удобен тем, что можно оперативно менять и настраивать стили на десятках страниц сайта, внося изменения в код всего одного файла.

Изображения тоже являются статическими файлами. Для хранения изображений мы ранее создали папку images. Разместим в этой папке файл с любым изображением и дадим ему имя image1.jpg. Теперь выведем его на нашей странице templates/firstapp/home.html, для чего модифицируем код страницы, как показано в листинге 6.27.

**Листинг 6.27. Фрагмент файла home.html**

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
```

```
<head>
    <meta charset="UTF-8">
    <title>Привет Django</title>
    <link href="{% static 'css/styles.css' %}" rel="stylesheet">
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
    
</body>
</html>
```

Здесь мы добавили всего одну строку, которая обеспечивает вывод изображения:

```

```

Снова обратимся к странице `home.html` и получим следующий результат (рис. 6.42).

Как можно видеть, изображение из папки `static/images/image1.jpg` успешно выведено на HTML-странице. Поскольку для изображений в файле `styles.css` не был указан стиль вывода, то оно отображается на странице в том виде, в каком было сохранено в файле

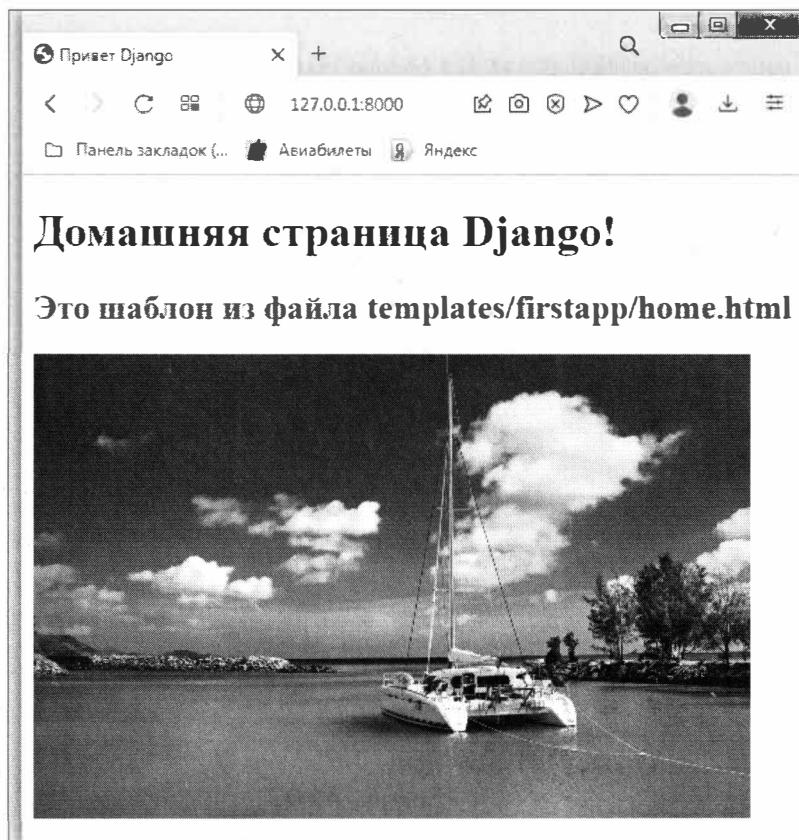


Рис. 6.42. Отображение на странице `home.html` текста в соответствии со стилями, взятыми из файла `styles.css`, и вывод рисунка из файла `image1.jpg`

image1.jpg. Однако для выводимых на HTML-страницах рисунков в файле styles.css тоже можно указывать стиль отображения. Изменим файл styles.css, как показано в листинге 6.28.

#### Листинг 6.28. Фрагмент файла styles.css

```
/* static/css/styles.css */  
body h1 {color: red;}  
body h2 {color: green;}  
img{width:250px;}
```

Здесь мы указали, что ширина изображения должна быть 250 пикселов. После такого изменения страница home.html будет выглядеть, как показано на рис. 6.43.

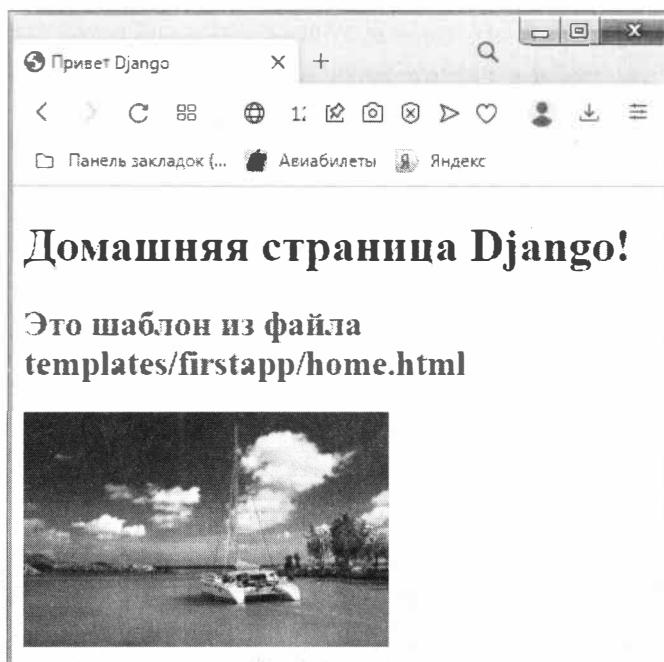


Рис. 6.43. Отображение на странице home.html текста в соответствии со стилями, взятыми из файла styles.css, и вывод рисунка из файла image1.jpg с использованием стиля для изображений

Как можно видеть из данного рисунка, то же изображение из файла image1.jpg на HTML-странице имеет другой размер, который соответствует стилю, указанному в файле styles.css.

## 6.9. Использование класса *TemplateView* для вызова шаблонов HTML-страниц

В примерах из предыдущих разделов, когда приходил запрос от браузера пользователя, система маршрутизации выбирала нужное представление (view), и уже оно вызывало

шаблон для генерации ответа. Кроме того, при необходимости представление может обратиться к БД и вставить в шаблон некоторые данные из нее. Однако если нужно просто возвратить пользователю содержимое шаблона, то для этого необязательно определять функцию в представлении и обращаться к ней. Можно воспользоваться встроенным классом `TemplateView` и вызвать нужный шаблон, минуя обращение к представлению.

На рис. 6.44 показан вариант приложения, в котором шаблон главной страницы сайта `home.html` вызывается функцией из представления `view`, а шаблоны страниц `about` и `contact` вызываются с использованием класса `TemplateView`.

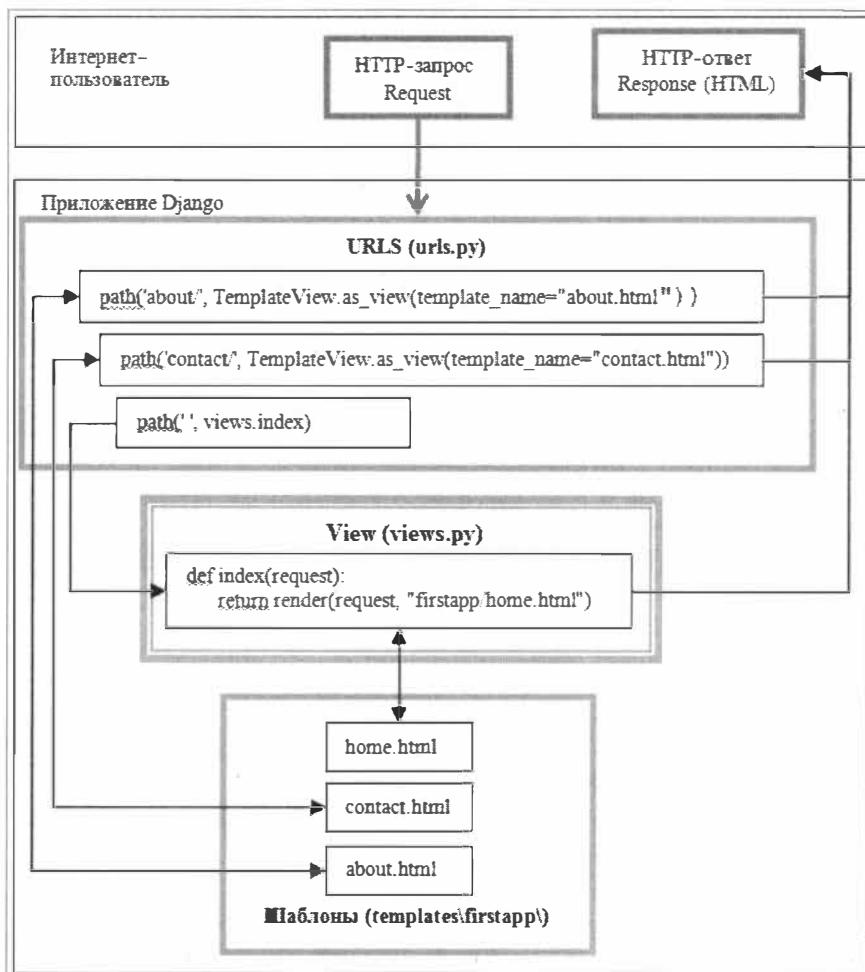


Рис. 6.44. Обращения к шаблонам HTML-страниц с использованием представления (view) и класса `TemplateView`

Проверим, как это работает на простых примерах, для этого создадим несколько простейших шаблонов в папке `hello\templates\firstapp\`. Пусть это будет файл-шаблон `about.html` с кодом из листинга 6.29.

**Листинг 6.29. Файл about.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1>Сведения о компании</h1>
    <h2>"Интеллектуальные информационные системы"</h2>
    <h3>Это шаблон из файла templates/firstapp/about.html</h3>
</body>
</html>
```

В этой же папке создадим файл-шаблон contact.html с кодом, приведенным в листинге 6.30.

**Листинг 6.30. Файл contact.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1>Контактные данные компании</h1>
    <h2>" Интеллектуальные информационные системы "</h2>
    <h3>Адрес: г. Москва, ул. Короленко, д. 24</h3>
    <h4> Это шаблон из файла templates/firstapp/contact.html</h4>
</body>
</html>
```

Осталось внести изменения в файл firstapp/urls.py (листинг 6.31).

**Листинг 6.31. Фрагмент файла urls.py**

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

urlpatterns = [
    path('', views.index),
    path('about/', TemplateView.as_view(template_name="firstapp/about.html")),
    path('contact/', TemplateView.as_view(
        template_name="firstapp/contact.html")),
]
```

В рассматриваемом программном коде сначала для вызова главной страницы сайта делается обращение к функции index() в представлении view, и уже функция index() вызывает главную страницу приложения firstapp/home.html:

```
def index(request):
    return render(request, "firstapp/home.html")
```

Затем в двух строках программного кода, приведенных в листинге 6.31, для вызова страниц `firstapp\about.html` и `firstapp\contact.html` используется класс `TemplateView`:

```
path('about/', TemplateView.as_view(template_name="firstapp/about.html")),
path('contact/', TemplateView.as_view(template_name="firstapp/contact.html")),
```

По своей сути класс `TemplateView` предоставляет функциональность представления. С помощью метода `as_view()` через параметр `template_name` задается путь к шаблону, который и будет возвращен в качестве ответа.

Запустим локальный веб-сервер и посмотрим содержимое всех трех страниц:

```
python manage.py runserver
```

Если все было сделано правильно, то мы получим следующий результат (рис. 6.45).

С помощью параметра `extra_context` в метод `as_view` можно передать данные для их отображения в шаблоне. Данные должны представлять собой словарь. В качестве примера передадим в шаблон `firstapp/contact.html` виды работ, которые выполняет компания. Для этого используем переменную `work` и изменим код файла `firstapp/urls.py`, как показано в листинге 6.32.

### Листинг 6.32. Фрагмент файла urls.py

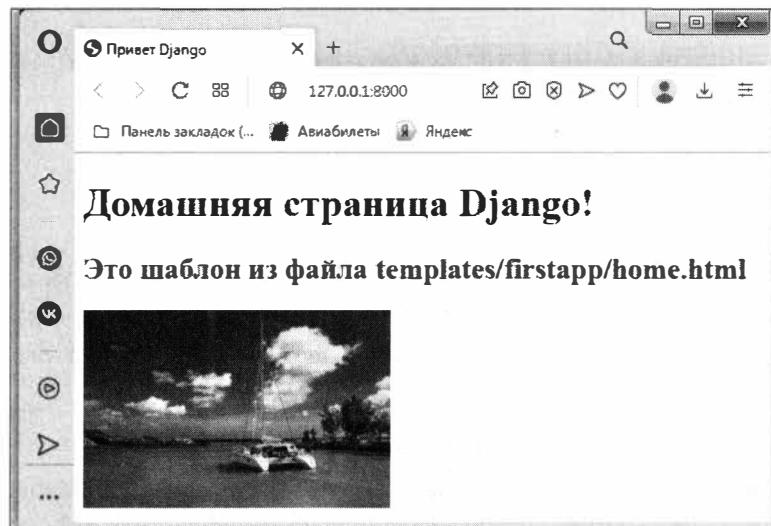
```
from django.urls import path
from .import views
from django.views.generic import TemplateView

urlpatterns = [
    path('', views.index),
    path('about/', TemplateView.as_view(template_name="firstapp/about.html")),
    path('contact/', TemplateView.as_view(template_name=
                                            "firstapp/contact.html",
                                            extra_context={"work":
                                                "Разработка программных продуктов"})),]
```

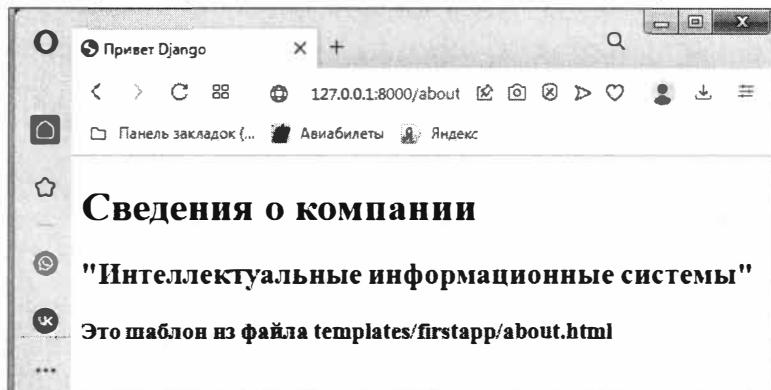
Здесь в шаблон `contact.html` передается объект `work`, который представляет строку со следующей информацией: "Разработка программных продуктов". И теперь мы можем использовать этот объект в шаблоне `firstapp/contact.html` (листинг 6.33).

### Листинг 6.33. Файл contact.html

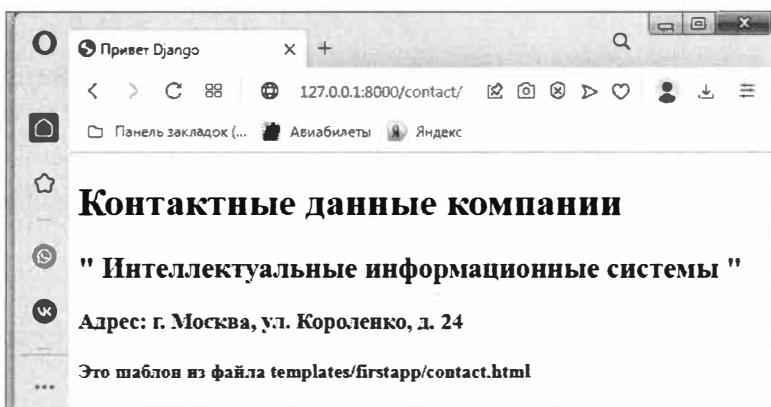
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет Django</title>
</head>
<body>
    <h1>Контактные данные компании</h1>
    <h2>"Интеллектуальные программные системы"</h2>
```



а



б



в

Рис. 6.45. Примеры обращения к шаблонам HTML-страниц с использованием представления view (а) и класса TemplateView (б и в)

```
<h3>{{ work }}</h3>
<h3>Адрес: г. Москва, ул. Короленко, д. 24</h3>
<h4>templates/firstapp/contact.html</h4>
</body>
</html>
```

В итоге получим следующий результат (рис. 6.46).

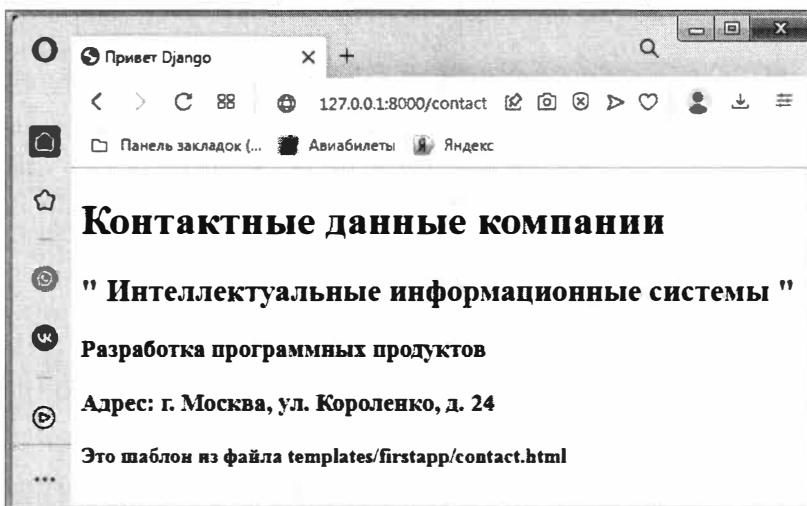


Рис. 6.46. Пример использования объекта work в шаблоне contact.html

## 6.10. Наследование шаблонов

Пожалуй, самая полезная функция Django — возможность наследования шаблонов. Наследование шаблонов позволяет создать базовый «скелетный» шаблон, который содержит все общие элементы сайта и определяет блоки, которые дочерние шаблоны могут переопределить.

Реализация наследования шаблонов основана на двух базовых тегах:

- `block` — создает блок, в который могут быть вставлены различные элементы HTML-страницы:
- `extend` — расширяет текущий шаблон.

Любой участок шаблона можно обернуть в тег `block`, при этом блоку дается имя, например:

```
{% block content %}  
    тело блока  
{% endblock %}
```

Любой родительский шаблон можно расширить путем создания на него ссылки из дочернего шаблона, например:

```
{% extend "base.htm" %}
```

В этой инструкции текущий шаблон расширяет базовый шаблон `base.htm`. Иными словами, при помощи тега `extend` мы указываем, какой шаблон мы будем расширять или уточнять.

Рассмотрим пример создания структуры сайта на основе базового шаблона и нескольких дочерних шаблонов HTML-страниц (рис. 6.47).

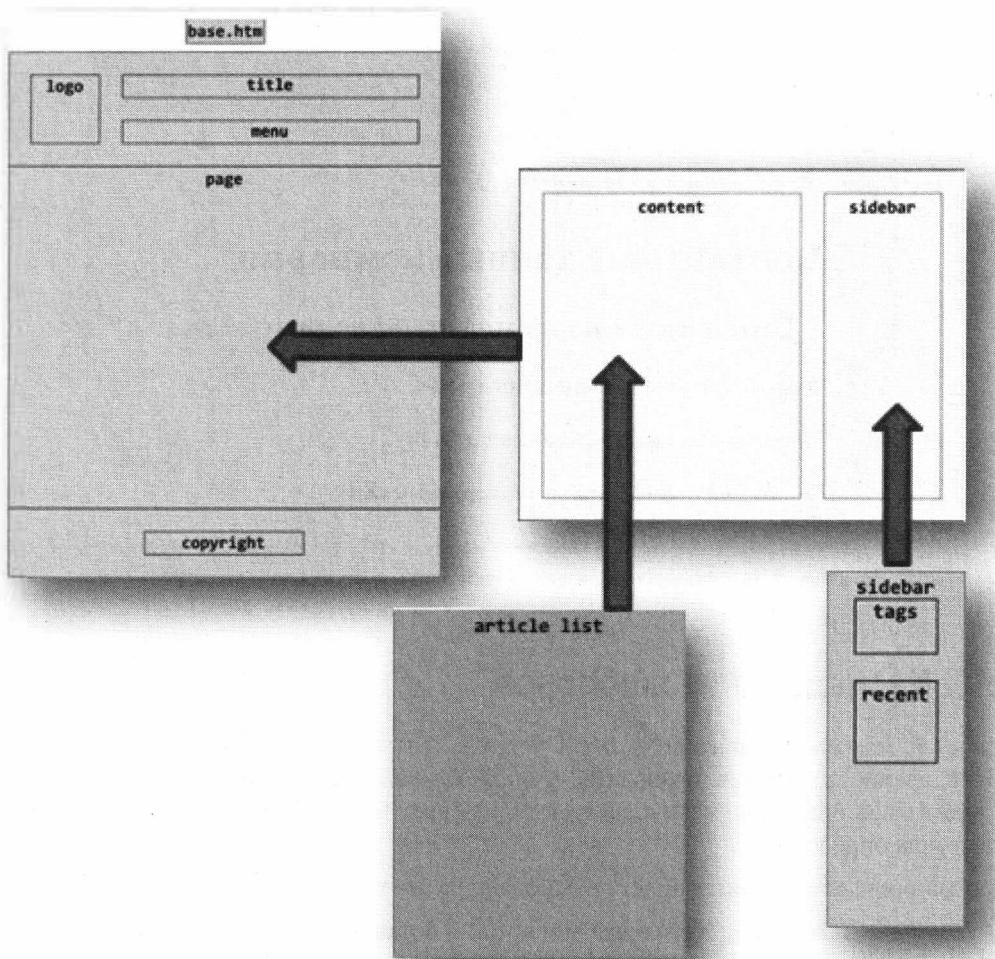


Рис. 6.47. Структура сайта на основе наследования шаблонов HTML-страниц

Как видно из данного рисунка, в структуре сайта есть базовый шаблон `base.html`, который может быть расширен за счет дочернего шаблона. В свою очередь, этот дочерний шаблон можно расширить за счет других шаблонов. Попробуем сформировать структуру сайта с использованием тегов `block` и `extend`.

Шаблон страницы `base.html` может иметь следующую структуру:

```
{% block head %}  
    {% block title %}{% endblock %}  
    {% block menu %}{% endblock %}  
{% endblock %}
```

```
{% block page %}
    {% block content %}
        {% endblock %}
    {% endblock %}

{% block footer %}
    {% block copyright %}
        {% endblock %}
    {% endblock %}
```

Созданные на основе этого кода блоки могут быть расположены на HTML-странице так, как показано на рис. 6.48.

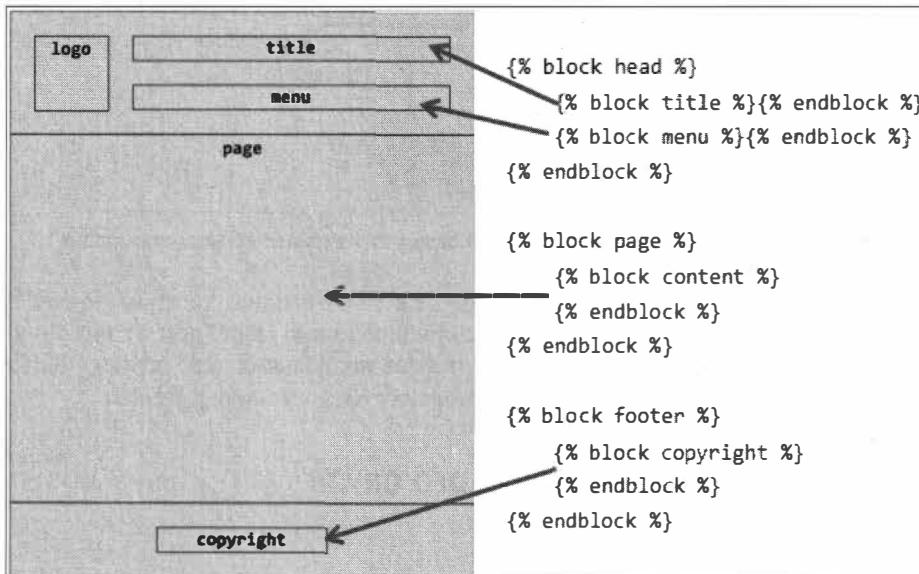


Рис. 6.48. Структура базового шаблона `base.html`

Этот базовый шаблон теперь может быть расширен на основе другого шаблона с помощью следующего кода:

```
{% extend "base.htm" %}
{% block page %}
    {% block content %}
    {% endblock %}
    {% block sidebar %}
    {% endblock %}
{% endblock %}
```

Здесь в первой строке указано, что это дочерний шаблон, который расширяет базовый шаблон с именем `base.html`:

```
{% extend "base.htm" %}
```

Следующие строки говорят о том, что в базовом шаблоне блок с именем `page`, будет заменен содержимым блока `page` из дочернего шаблона (рис. 6.49).

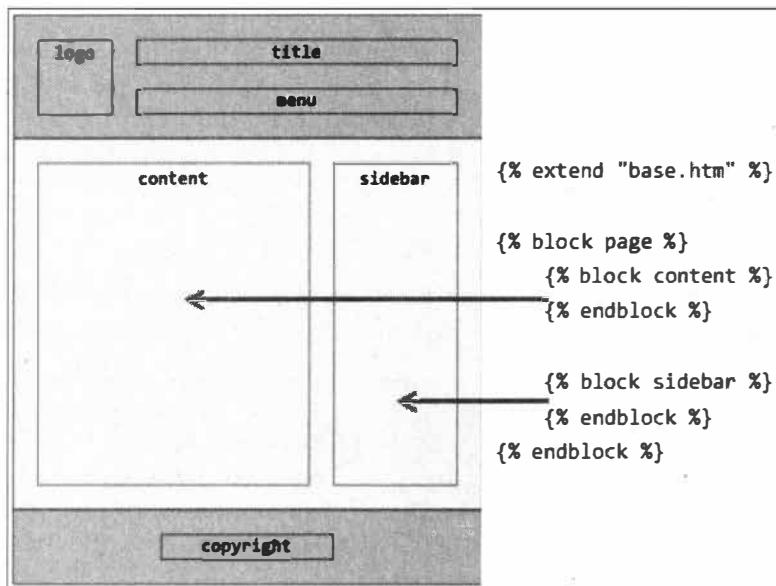


Рис. 6.49. Расширение базового шаблона base.html на основе дочернего шаблона

Эту цепочку наследования можно продолжать сколько угодно, каждым новым дочерним шаблоном расширять один из родительских шаблонов. Перейдем от теории к практике и создадим сначала фрагмент сайта на основе нескольких «скелетных» шаблонов, а потом «украсим» эти шаблоны за счет возможностей фреймворка Bootstrap.

## 6.11. Создание многостраничного сайта на основе шаблонов Django

Как мы уже знаем, в Django HTML-страницы веб-приложения не являются статическими, это динамические страницы, которые строятся на основе шаблонов. Шаблоны в процессе работы приложения заполняются определенной информацией, которая, как правило, загружается из базы данных. Следует различать два типа шаблонов: базовый шаблон приложения и шаблоны динамически подгружаемых страниц. Базовый шаблон является основой всего веб-приложения, на нем обычно располагаются следующие элементы приложения:

- верхняя строка или заголовок сайта с логотипом компании и главным меню сайта;
- боковая панель с дополнительным меню сайта или с опциями фильтрации содержания других страниц;
- центральная панель или область, в которой будут отображаться все прочие страницы сайта, т. е. основной контент;
- нижняя панель (footer), или «подвал», на котором отображается информация о владельцах или создателях сайта, а также ссылки на дополнительные информационные ресурсы.

Как правило, заголовок сайта, боковая панель и «подвал» всегда присутствуют в видимом окне приложения, а вот содержимое центральной области (контента) постоянно

меняется в зависимости от того, какая страница сайта была загружена в текущий момент. Схематично взаимодействие шаблонов приведено на рис. 6.50.

Здесь `base.html` — базовый (родительский) шаблон, остальные страницы сайта — это дочерние элементы, которые будут своим содержимым дополнять блок с контентом базового шаблона. Базовый шаблон может иметь другую структуру и иной набор элементов, например, может присутствовать «карусель» в верхней части страницы, правая боковая панель, группа раскрывающихся элементов типа «аккордеон» и т. п.

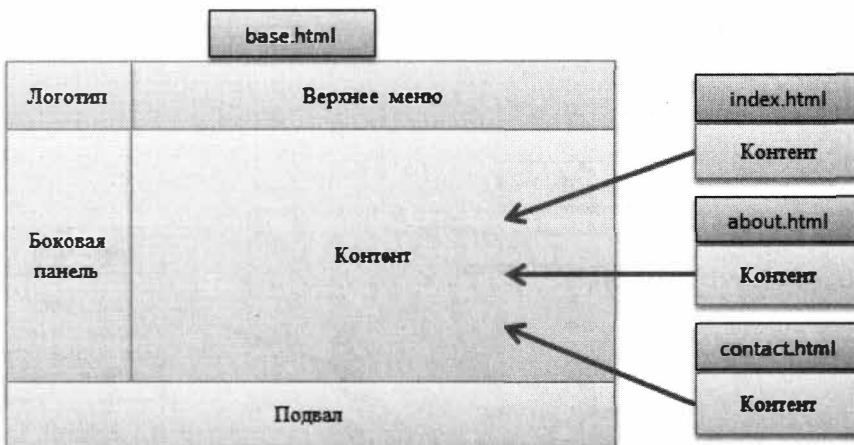


Рис. 6.50. Схема взаимодействия шаблонов в Django

Хорошим тоном программирования считается формирование единообразного стиля сайта, когда веб-страницы имеют одни и те же структурные элементы: заголовок сайта (`header`), меню, подвал (`footer`), боковую панель (`sidebar`) и т. д. А вот блок главной страницы с контентом может постоянно меняться.

Конечно, можно сконфигурировать каждую страницу сайта как самостоятельную и независимую единицу. Однако если возникнет необходимость изменить какой-то блок, например добавить в общее меню еще один пункт, то придется менять все страницы, которых может быть довольно много. В этом случае более рационально многократно использовать один базовый шаблон, который определяет все основные блоки страниц сайта. Тогда все остальные шаблоны страниц будут иметь одинаковую базовую структуру и одни и те же блоки, при этом в отдельных блоках можно менять содержимое.

В качестве примера создадим достаточно простую структуру сайта, который будет иметь один базовый шаблон и три дочерние страницы (рис. 6.51).

У данного сайта будет базовый шаблон (`base.html`) и три страницы расширения базового шаблона: главная страница сайта (`index.html`), о компании (`about.html`) и контакты компании (`contact.html`). Для начала в папке `templates/firstapp/` создадим базовый шаблон, который назовем `base.html`, и напишем в нем код, приведенный в листинге 6.34.

#### Листинг 6.34. Файл `base.html`

```
<!DOCTYPE html>
<html lang="en">
```

```

<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock title %}</title>
</head>
<body>
    <h1>Логотип и Заголовок сайта</h1>
    <h4>Главная страница О компании Контакты</h4>
    <div>{% block header%}{% endblock header %}</div>
    <div>{% block content%}{% endblock content %}</div>
    <h5>Подвал (footer) сайта</h5>
</body>
</html>

```

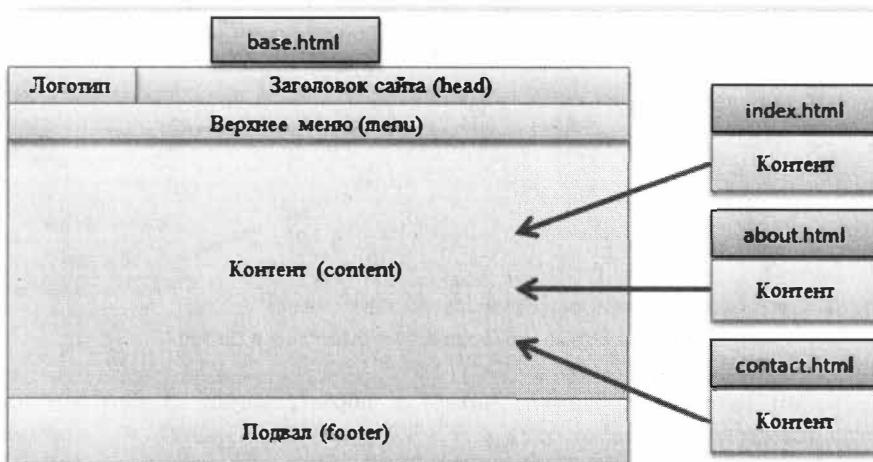


Рис. 6.51. Структура сайта из базового шаблона и трех страниц

Здесь с помощью элементов

```
{% block название_блока %}{% endblock название_блока %}
```

определяются отдельные блоки шаблонов. При этом для каждого блока задается открывающий элемент:

```
{% block название_блока %}
```

и закрывающий элемент:

```
{% endblock название_блока %}
```

Например, блок `title` имеет такую структуру:

```
<title>{% block title %}{% endblock title %}</title>
```

Когда другие шаблоны будут использовать этот базовый шаблон, то они могут вставить в блок `title` новое содержимое. Подобным же образом здесь определены блоки `header` и `content`:

```
<div>{% block header%}{% endblock header %}</div>
<div>{% block content%}{% endblock content %}</div>
```

Для каждого блока можно задать содержимое по умолчанию, самих же блоков в шаблонах может быть сколько угодно.

В базовом шаблоне также определены следующие неизменяемые элементы: логотип компании и заголовок сайта, главное меню, подвал страницы (footer). Поскольку эти элементы будут общими для всех страниц, то мы не обрамляем их блоками, а задаем им значения в виде констант.

Теперь сформируем дочернюю страницу и расширим наш базовый шаблон. Создадим в каталоге `templates\firstapp` новый файл-шаблон главной страницы сайта с именем `index.html` и напишем в нем код из листинга 6.35.

#### Листинг 6.35. Фрагмент файла `index.html`

```
{% extends "firstapp/base.html" %}  
{% block title %}Главная страница{% endblock title %}  
{% block header %}Это главная страница сайта - index.html.{% endblock header %}  
{% block content%}  
    Это содержимое блока content. Этот текст находится на странице index.html,  
    но он вставлен в базовый шаблон - base.html в блок с именем context.  
{% endblock content %}
```

Здесь с помощью выражения `{% extends "firstapp/base.html" %}` мы определили, что эта страница будет расширять базовый шаблон с именем `base.html`, который находится в каталоге шаблонов приложения `firstapp` (по пути: `firstapp/base.html`). Затем задали содержимое для блоков `title`, `header` и `content`. Содержимое этих блоков будет передано в базовый шаблон (`base.html`).

Изменим код функции `def index ()` в представлении `view`, т. е. содержимое файла `firstapp/views.py` (листинг 6.36).

#### Листинг 6.36. Фрагмент файла `views.py`

```
def index(request):  
    return render(request, "firstapp/index.html")
```

Здесь в функции `index()` в качестве домашней страницы сайта указан файл `firstapp/index.html`.

Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на главную страницу `http://127.0.0.1:8000/`, то мы увидим следующий результат (рис. 6.52).

Как видно из данного рисунка, страница `index.html` является дочерней страницей базового шаблона. При этом постоянные элементы базового шаблона не изменились, а в блоки `title`, `header` и `content` был вставлен текст из соответствующих блоков дочернего шаблона. Обратите внимание, что в коде страницы `index.html` нет ни одного фрагмента, написанного на языке HTML, при этом содержание этой страницы успешно выведено в окно браузера.

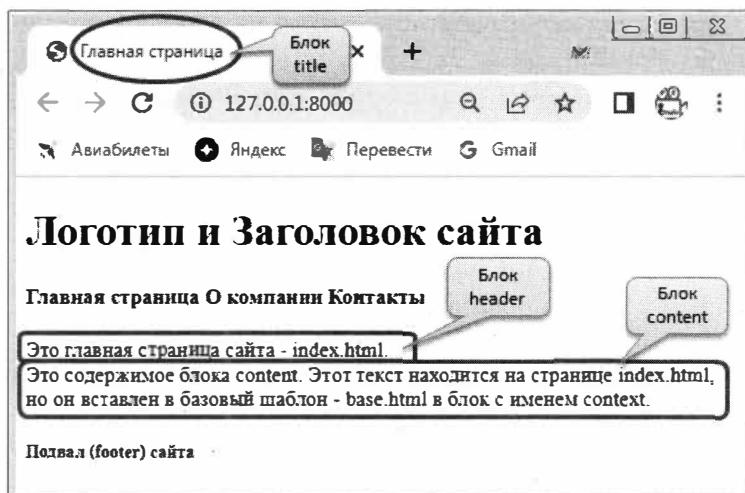


Рис. 6.52. Главная страница сайта index.html, которая является расширением базового шаблона base.html

Теперь для того же базового шаблона base.html сформируем другую страницу сайта — firstapp/about.html. Эта страница уже существует, т. к. она была создана в предыдущих разделах. Удалим содержимое этой страницы и напишем в ней программный код, приведенный в листинге 6.37.

#### Листинг 6.37. Файл about.html

```
% extends "firstapp/base.html" %
% block title %} О компании{%
% endblock title %
% block header %}Сведения о компании{%
% endblock header %
% block content %
    Компания "Интеллектуальные информационные системы".
    Наша компания занимается разработкой приложений,
    основанных на системах компьютерного зрения
    и искусственного интеллекта, кроссплатформенных настольных
    мобильных приложений, а также веб-приложений
    с использованием языка программирования Python.
    Это содержимое блока content. Этот текст находится на
    странице about.html, но он вставлен в базовый
    шаблон - base.html, в блок с именем context.
% endblock content %}
```

Здесь с помощью выражения `{% extends "firstapp/base.html" %}` мы определили, что эта страница будет расширением базового шаблона с именем `base.html`, который находится в каталоге шаблонов приложения `firstapp` (по пути: `firstapp\base.html`). Затем задали содержимое для блоков `title`, `header` и `content`. Эти значения и будут переданы в базовый шаблон (`base.html`).

Следующим шагом нужно открыть файл `firstapp\urls.py` и добавить в него URL-адрес страницы `about.html`, т. е. строку:

```
path('about/', views.about),
```

Теперь файл firstapp/urls.py должен выглядеть так, как показано в листинге 6.38.

#### Листинг 6.38. Фрагмент файла urls.py

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index),
    path('about/', views.about),
]
```

Далее нужно в файле firstapp/views.py создать функцию about(). Программный код файла firstapp/views.py к данному моменту должен выглядеть так, как показано в листинге 6.39.

#### Листинг 6.39. Фрагмент файла views.py

```
from django.shortcuts import render
def index(request):
    return render(request, "firstapp/index.html")
def about(request):
    return render(request, "firstapp/about.html")
```

Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на страницу <http://127.0.0.1:8000/about/>, то мы увидим следующий результат (рис. 6.53).

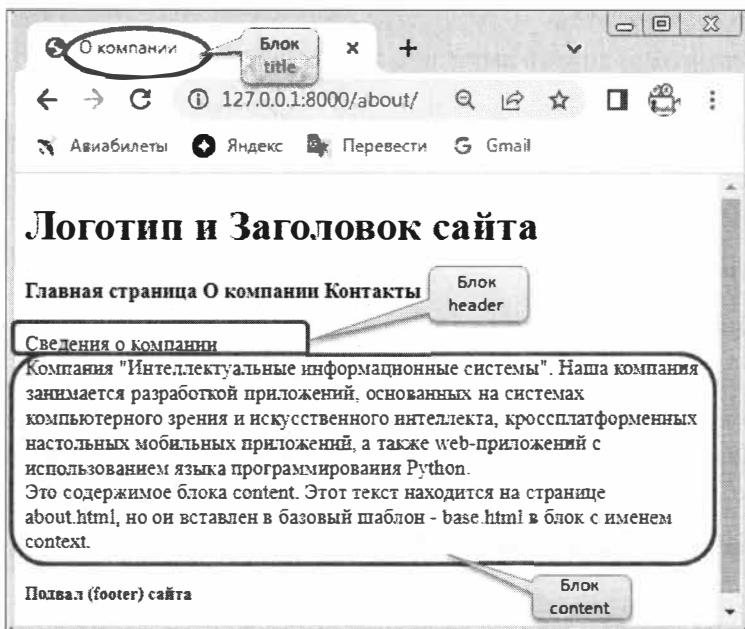


Рис. 6.53. Страница сайта about.html, которая является расширением базового шаблона base.html

Теперь перейдем к формированию третьей страницы нашего упрощенного сайта — firstapp/contact.html. Эта страница уже существует, т. к. она была создана в предыдущих разделах. Удалим содержимое этой страницы и напишем в ней программный код из листинга 6.40.

#### Листинг 6.40. Файл contact.html

```
% extends "firstapp/base.html"
% block title %}Контакты{%
% block header %}Контактные данные.{%
% block content %
    Компания "Интеллектуальные информационные системы".
    Адрес: г. Москва, ул. Короленко, д. 24
    тел.: +7 999 999 99
    e-mail: iis@yandex.ru
    Это содержимое блока content. Этот текст находится
    на странице contact.html, но он вставлен в базовый
    шаблон - base.html, в блок с именем context.
% endblock content %}
```

Здесь с помощью выражения `% extends "firstapp/base.html"` мы определили, что эта страница будет расширением базового шаблона с именем `base.html`, который находится в каталоге шаблонов приложения `firstapp` (по пути: `firstapp\base.html`). Затем задали содержимое для блоков `title`, `header` и `content`. Эти значения и будут переданы в базовый шаблон (`base.html`).

Следующим шагом нужно открыть файл `firstapp/urls.py` и добавить в него URL-адрес страницы `contact.html`, т. е. строку

```
path('contact/', views.contact),
```

Теперь файл `firstapp/urls.py` должен выглядеть так, как показано в листинге 6.41.

#### Листинг 6.41. Фрагмент файла urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index),
    path('about/', views.about),
    path('contact/', views.contact),
]
```

Теперь нужно в файле `firstapp/views.py` создать функцию `contact()`. Программный код файла `firstapp/views.py` к данному моменту должен выглядеть так, как показано в листинге 6.42.

#### Листинг 6.42. Фрагмент файла views.py

```
from django.shortcuts import render
def index(request):
    return render(request, "firstapp/index.html")
```

```
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
```

Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на страницу <http://127.0.0.1:8000/contact/>, то мы увидим следующий результат (рис. 6.54).

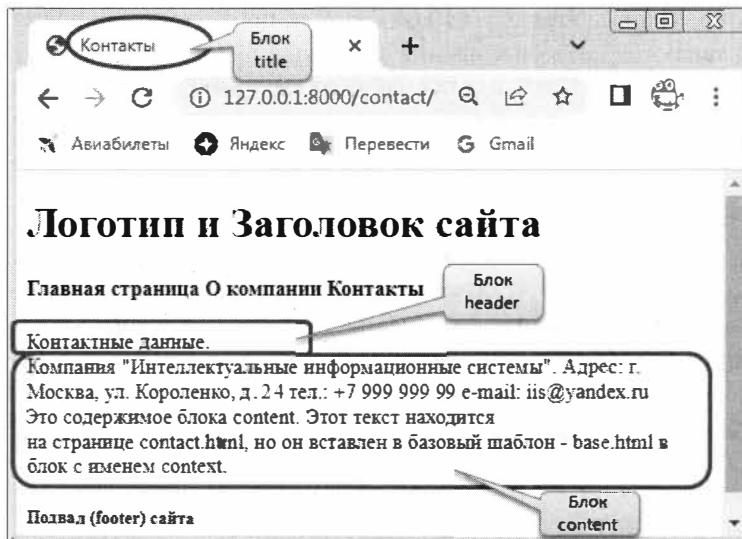


Рис. 6.54. Страница сайта contact.html, которая является расширением базового шаблона base.html

Итак, мы создали базовый шаблон и три страницы сайта, которые являются расширением базового шаблона. Подводя итоги этого раздела, отметим, что на основе базового шаблона можно создавать разные страницы сайта, с общими элементами и блоками, в которых можно размещать разную информацию.

В базовом шаблоне есть меню, состоящее из трех опций: «Главная страница», «О компании» и «Контакты». Однако в меню не прописаны ссылки для того, чтобы по ним можно было перейти на соответствующие страницы. Давайте рассмотрим, как в шаблонах Django реализована адресация на другие шаблоны (фактически на другие веб-страницы).

## 6.12. Формирование URL-адресов в шаблонах Django

На данный момент мы научились создавать шаблоны, прописывать в них различные теги `{% имя_тега %}`, переменные `{{ имя_переменной }}` и фильтры `{{ value|имя_фильтра }}`. Следующий важный шаг — научиться указывать URL-адреса в шаблонах.

В шаблонах, как и в традиционных HTML-страницах, для этого есть специальный тег:

```
<a href= "url">...</a>
```

Однако проставлять прямые ссылки на страницы сайта — это не лучший вариант. Сегодня ваш сайт находится в разработке на вашем компьютере по адресу: 127.0.0.1:800, завтра он будет развернут на сервере по адресу (доменному имени): mysite.ru. Еслиставить полные пути до страниц (например, 127.0.0.1/catalog/books), то при каждом переезде сайта придется менять все ссылки во всем проекте. Чтобы избежать таких ситуаций, в шаблонах Django для создания ссылок предусмотрен специальный тег, который имеет следующий синтаксис:

```
{% url 'url-name' v1 v2 %}
```

Здесь первый аргумент — имя шаблона URL (*url-name*). Это может быть заключенный в кавычки литерал или любая другая контекстная переменная. Дополнительные аргументы (*v1*, *v2*) являются необязательными и должны представлять собой значения, разделенные пробелами, которые будут использоваться в качестве аргументов в URL.

Имена ссылок '*url-name*' можно указать в файле *urls.py*. Они передаются как параметр *name* при создании пути (*path*), например:

```
path('', views.index, name='index')
```

Вернемся к нашему сайту, созданному в предыдущем разделе, и в строке, которая содержит меню, создадим соответствующие ссылки. Начнем с файла *firstapp/urls.py*, где каждому URL-адресу присвоим имя. Изменения, внесенные в код файла *firstapp/urls.py*, приведены в листинге 6.43 (выделены серым фоном).

#### Листинг 6.43. Фрагмент файла urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Как видно из данного листинга, главной странице сайта задано имя *name='index'*, странице «О компании» — *name='about'*, странице «Контакты» — *name='contact'*.

Теперь изменим базовый шаблон (файл *templates/firstapp/base.html*). Изменения, внесенные в код этого файла, приведены в листинге 6.44 (выделены серым фоном).

#### Листинг 6.44. Файл base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}{% endblock title %}</title>
</head>
<body>
    <h1>Логотип и Заголовок сайта</h1>
```

```

<h4>
    <a href="{% url 'index' %}">Главная страница</a>
    <a href="{% url 'about' %}">О компании</a>
    <a href="{% url 'contact' %}">Контакты</a>
</h4>
<div>{% block header%}{% endblock header %}</div>
<div>{% block content%}{% endblock content %}</div>
<h5>Подвал (footer) сайта</h5>
</body>
</html>

```

В этом программном коде была изменена всего одна строка:

```
<h4>Главная страница О компании Контакты</h4>
```

Эта строка была разбита на три строки и с использованием тега `{% url 'name' %}` были созданы ссылки на шаблоны страниц: index (Главная страница), about (О компании) и contact (Контакты). После этих изменений переходить между страницами сайта можно через меню, которое находится в базовом шаблоне.

Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на главную страницу `http://127.0.0.1:8000/`, то мы увидим следующий результат (рис. 6.55).

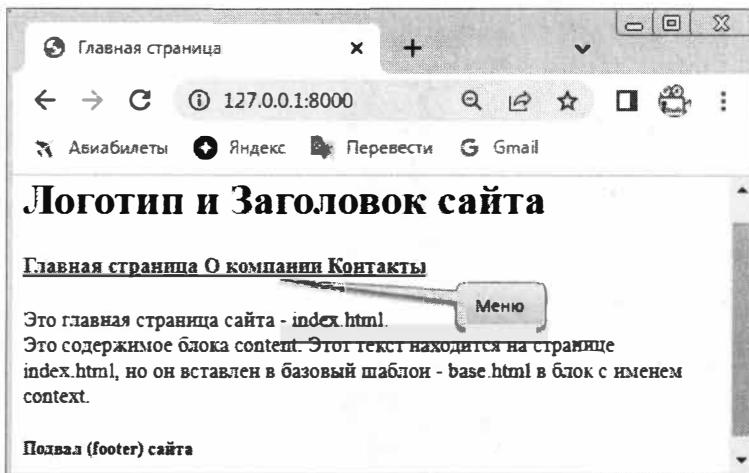


Рис. 6.55. Главная страница сайта с меню, которое находится в шаблоне base.html

Как видно из данного рисунка, на странице появилось меню. Каждая опция меню выделена подчеркиванием и синим цветом. Теперь переключаться между страницами сайта можно через меню, которое присутствует на каждой странице, при этом оно фактически размещено в базовом шаблоне.

Итак, с помощью шаблонов мы создали простой сайт, состоящий из трех страниц. Шаблоны Django позволяют достаточно быстро и просто формировать технологическую часть веб-приложения, но, если вы обратили внимание, все страницы выполнены

в черно-белом цвете и им не хватает красок. Однако фреймворк Django не предназначен для этого, он служит для упрощения процессов создания серверной части веб-приложений (backend-разработка), а для создания ярких привлекательных HTML-страниц применяется другой инструментарий: CSS и Java-скрипты. С созданием привлекательного пользовательского интерфейса успешно справляется другой фреймворк — Bootstrap, который достаточно просто интегрируется с Django. Воспользуемся возможностями Bootstrap и внесем ярких красок в только что созданный нами сайт.

## 6.13. Интеграция шаблонов Django с фреймворком Bootstrap

В главе 3 мы познакомились с основами макетирования HTML-страниц с использованием фреймворка Bootstrap, теперь применим эти знания на практике. Немного изменим код страниц сайта, созданного в предыдущем разделе, добавив в него красок. Для начала добавим папки `css` и `js` к нашему проекту. Напомним, что скачать готовые файлы фреймворка Bootstrap можно с официального сайта по следующей ссылке:

<https://getbootstrap.com/docs/5.2/getting-started/download/>.

После скачивания и распаковки архива видно, что все файлы Bootstrap находятся в двух папках: `css` и `js`. Скопируем эти файлы в наш проект в папку `hello/static`. Расположение этих папок в структуре нашего проекта показано на рис. 6.56.

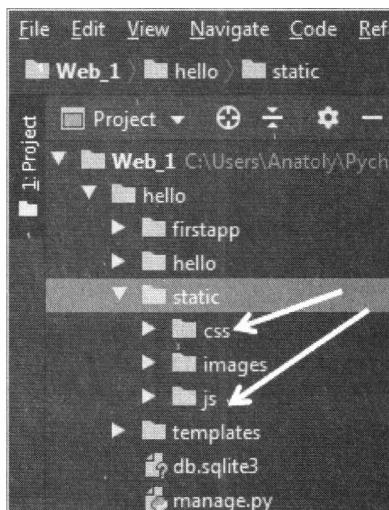


Рис. 6.56. Расположение папок с файлами Bootstrap в структуре проекта

Напомним, что в Bootstrap предусмотрены контейнеры, в которые вкладываются строки, в строках вкладываются колонки, из которых формируются адаптивные блоки, а для формирования макета страниц используется тег `<div>`.

Можно подключить фреймворк Bootstrap к проекту Django и другим способом. Загрузить в приложение пакет `django-bootstrap-v5` командой

```
pip install django-bootstrap-v5
```

В файле `settings.py` в переменную `INSTALLED_APPS` добавить строку (листинг 6.45).

**Листинг 6.45. Фрагмент файла settings.py**

```
INSTALLED_APPS = (
    # ...
    "bootstrap5",
    # ...
)
```

В шаблонах загрузить библиотеку bootstrap5 с использованием следующего тега:

```
{% load bootstrap5 %}
```

Но мы воспользуемся первым вариантом и просто добавим файлы Bootstrap в папку static.

Итак, чтобы внести яркость красок в наш сайт, достаточно изменить базовый шаблон. Откроем файл hello/templates/firstapp/base.html и модифицируем его, как показано в листинге 6.46 (строки с подключением Bootstrap выделены серым фоном).

**Листинг 6.46. Файл base.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
    <title>{% block title %}{% endblock title %}</title>
</head>
<body>
    <div class="container-fluid p-1 bg-primary text-white text-center">
        <div class="row">
            <div class="col-2 ">
                
            </div>
            <div class="col-10 ">
                <h1>Это заголовок главной страницы сайта</h1>
            </div>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row bg-warning text-center">
            <h6>
                <a href="{% url 'index' %}">Главная страница</a>
                <a href="{% url 'about' %}">О компании</a>
                <a href="{% url 'contact' %}">Контакты</a>
            </h6>
        </div>
    </div>
</div>
```

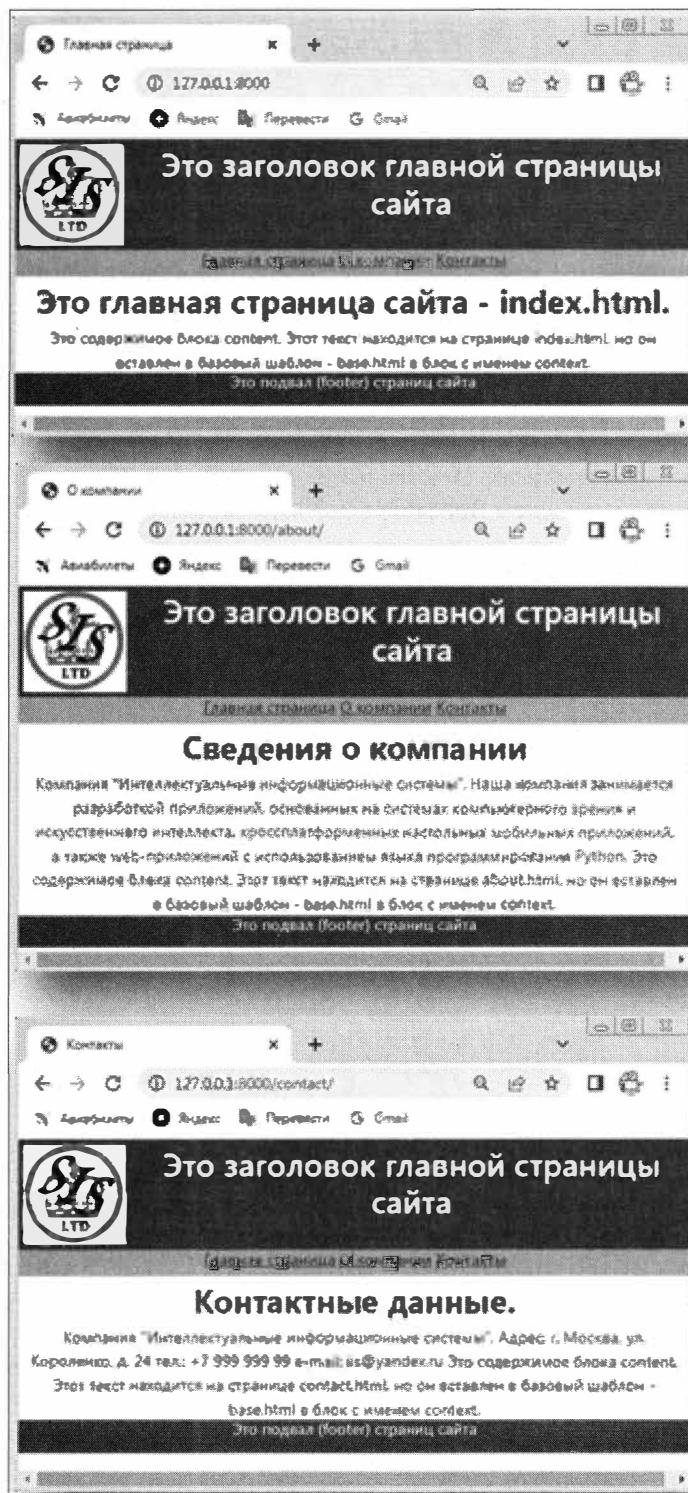


Рис. 6.57. Страницы сайта после подключения в базовый шаблон тегов Bootstrap

```
<div class="container-fluid">
    <div class="row text-center text-primary fs-1 fw-bold">
        <div>{% block header%}{% endblock header %}</div>
    </div>
    <div class="row text-center text-body">
        <div>{% block content%}{% endblock content %}</div>
    </div>
</div>
<div class="container-fluid">
    <div class="row bg-primary text-center text-white lh-1">
        <p>Это подвал (footer) страниц сайта</p>
    </div>
</div>
</body>
</html>
```

Далее созданы четыре контейнера для размещения следующих компонентов:

- изображения с логотипом компании и заголовка сайта;
- главного меню;
- блоков `header` и `content`;
- подвала (`footer`).

Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на главную страницу <http://127.0.0.1:8000/>, то мы увидим следующий результат (рис. 6.57).

Обратите внимание, как сильно изменился внешний вид всех страниц сайта, хотя изменения были внесены только в базовый шаблон. При этом было выполнено только макетирование страниц на основе классов Bootstrap и не были задействованы специальные элементы пользовательского интерфейса.

Посмотрим теперь, как можно использовать некоторые специальные теги Django в комбинации с фреймворком Bootstrap.

## 6.14. Использование специальных тегов в шаблонах Django

В Django предоставлена возможность использовать в шаблонах ряд специальных тегов, которые упрощают вывод некоторых данных. Рассмотрим наиболее распространенные теги.

### 6.14.1. Тег для вывода текущей даты и времени

Для вывода дат в Django предусмотрен следующий тег:

```
{% now "формат_данных" %}
```

Тег `now` позволяет вывести системное время и дату. В качестве параметра тегу `now` передается формат данных, который указывает, как форматировать время и дату. Символы для форматирования времени и даты приведены в табл. 6.1.

Все возможные форматы для вывода даты и времени можно посмотреть в оригинальной документации на Django.

**Таблица 6.1. Символы для форматирования даты и времени**

№ п/п	Символ	Значение даты и времени
1	Y	Год в виде четырех цифр (2021)
2	y	Год в виде последних двух цифр (21)
3	F	Полное название месяца (Июль)
4	M	Сокращенное название месяца — 3 символа (Июл)
5	m	Номер месяца — две цифры (07)
6	N	Аббревиатура месяца в стиле Ассошиэйтед Пресс
7	n	Номер месяца — одна цифра (7)
8	j	День месяца (1-31)
9	l	День недели — текст (среда)
10	h	Часы (0-12) — 9:15
11	H	Часы (0-24) — 21:15
12	i	Минуты (0-59)
13	s	Секунды (0-59)

Изменим код шаблона страницы `templates/firstapp/about.html`, которая была создана в предыдущем разделе, как показано в листинге 6.47.

#### Листинг 6.47. Файл `about.html`

```
% extends "firstapp/base.html" %
% block title %}О компании{%
% block header %}Специальные теги{%
% block content %
<p>Примеры вывода даты и времени</p>
<div class="container-fluid text-start">
<li>Год в виде четырех цифр - {%
    now "Y" %}</li>
<li>Год в виде последних двух цифр - {%
    now "y" %}</li>
<li>Полное название месяца - {%
    now "F" %}</li>
<li>Сокращенное название месяца - {%
    now "M" %}</li>
<li>Номер месяца, две цифры - {%
    now "m" %}</li>
<li>Аббревиатура месяца - {%
    now "N" %}</li>
<li>Номер месяца, одна цифра - {%
    now "n" %}</li>
<li>День месяца - {%
    now "j" %}</li>
<li>День недели - текст {%
    now "l" %}</li>
```

```
<li>Часы (0-12) - { % now "h" %}</li>
<li>Часы (0-24) - { % now "H" %}</li>
<li>Минуты (0-59) - { % now "i" %}</li>
<li>Секунды (0-59) - { % now "s" %}</li>
<li>Дата (день/месяц/год) - { % now "j/m/Y" %}</li>
<li>Время (час:мин:сек) - { % now "H:i:s" %}</li>
</div>
{ % endblock content %}
```

Если обратиться к странице `about.html` после этих изменений, то мы получим следующий результат (рис. 6.58).

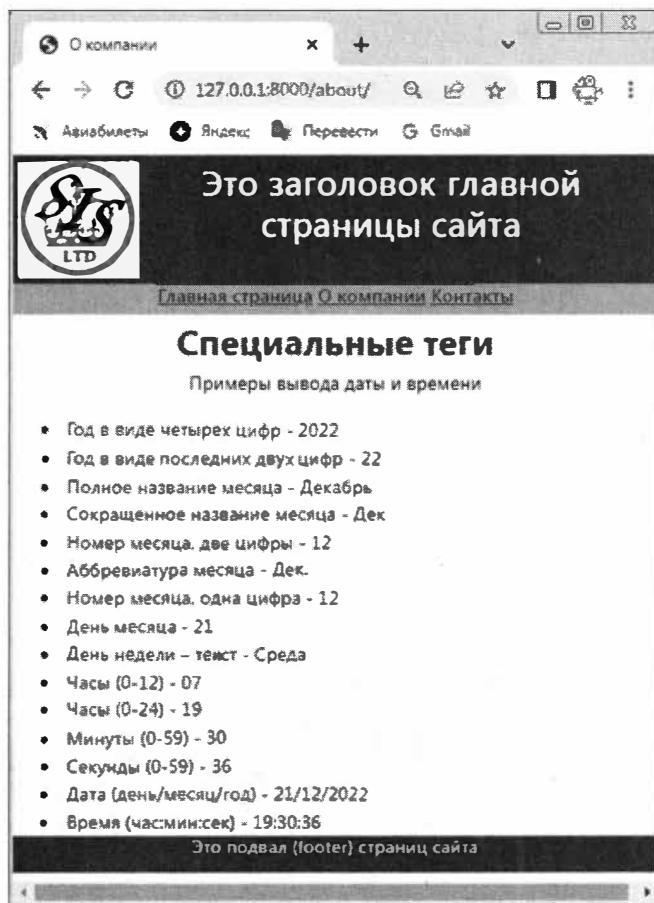


Рис. 6.58. Страница сайта `about.html` (варианты вывода текущей даты и времени)

## 6.14.2. Теги Bootstrap для вывода информации в адаптивных блоках

Если посмотреть на страницу «Контакты», то в ней реквизиты компании не упорядочены. Исправим эту ситуацию, для чего воспользуемся адаптивными блоками Bootstrap.

Изменим код шаблона страницы `templates/firstapp/contact.html`, которая была создана в предыдущем разделе, как показано в листинге 6.48.

#### Листинг 6.48. Файл contact.html

```
{% extends "firstapp/base.html" %}  

{% block title %}Контакты{% endblock title %}  

{% block header %}Контактные данные{% endblock header %}  

{% block content %}  

    Компания "Интеллектуальные информационные системы"  

<div class="container-fluid text-center my-2 border border-3">  

    <div class="row ">  

        <div class="col-sm-6 ">  

            <h3>Адрес</h3>  

            <p>г. Москва, ул. Короленко, д. 24</p>  

        </div>  

        <div class="col-sm-3 ">  

            <h3>Телефон</h3>  

            <p>+7 999 999 99</p>  

        </div>  

        <div class="col-sm-3 ">  

            <h3>e-mail</h3>  

            <p>iis@yandex.ru</p>  

        </div>  

    </div>  

</div>  

{% endblock content %}
```

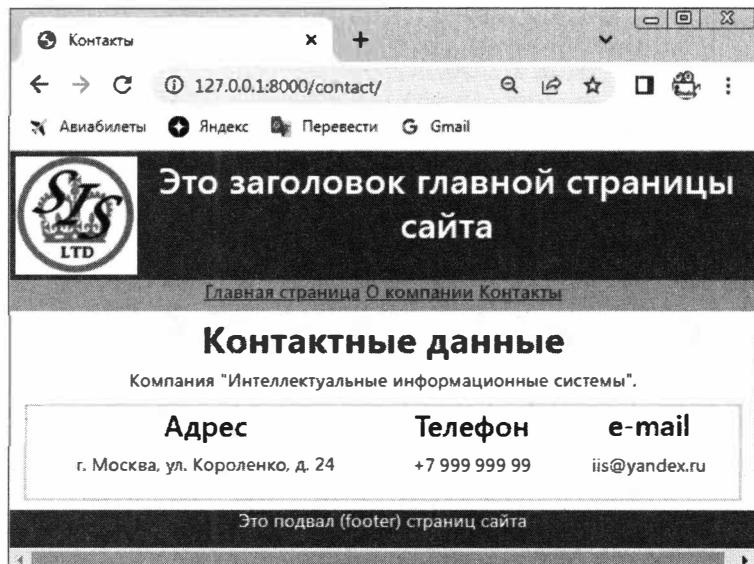


Рис. 6.59. Страница сайта `contact.html` (использование Bootstrap для макетирования шаблона страницы)

В этом программном коде мы использовали теги Bootstrap для макетирования страницы. Был создан контейнер, в который поместили один ряд, а в ряду выделили три колонки. В первой колонке указали адрес, во второй — телефон, в третьей — e-mail.

Если обратиться к странице contact.html после этих изменений, то мы получим следующий результат (рис. 6.59).

Как видно из рис. 6.59, контактные данные упорядочены и легко читаются. В этом случае теги Bootstrap использовались не в базовом шаблоне, а в шаблоне дочерней страницы contact.html.

### 6.14.3. Теги Bootstrap и Django для представления списков в виде таблицы

У фреймворка Bootstrap есть специальные теги для создания таблиц, тем не менее можно длинные списки представлять в виде таблиц с использованием тегов `<div>` и тегов Django для организации циклов. Предположим, что у нас есть список месяцев года и нам нужно вывести их на страницу сайта в разрезе каждого квартала. Пример такого шаблона приведен в листинге 6.49.

#### Листинг 6.49. Шаблон главной страницы сайта index.html

```
{% extends "firstapp/base.html" %}  
{% block title %}Главная страница{% endblock title %}  
{% block header %}Это главная страница сайта - index.html.{% endblock header %}  
{% block content %}  
    Месяцы (поквартально)  
    <div class="container-fluid my-2 text-start">  
        <div class="row">  
            <div class="col-6 text-end">  
                {% for kv in my_kv %}  
                    <div>{{ kv }}</div>  
                {% endfor %}  
            </div>  
            <div class="col-6">  
                <div class="row">  
                    {% for month in my_month %}  
                        <div class="col-4">{{ month }}</div>  
                    {% endfor %}  
                </div>  
            </div>  
        </div>  
    </div>  
{% endblock content %}
```

Здесь создан контейнер, в который вложена одна строка, разбитая на две колонки. В первой колонке с помощью тега организации цикла `{% for kv in my_kv %}` выводятся четыре строки названий кварталов, а во второй колонке за счет тега организации цикла `{% for month in my_month %}` выводятся четыре строки с названиями месяцев каждого квартала. Причем длинный список месяцев будет автоматически разбит на четыре строки благодаря тегу `<div class="col-4">{{ month }}</div>` фреймворка Bootstrap.

Теперь в этот шаблон нужно передать список с названиями кварталов (`my_kv`) и список с названиями месяцев (`my_month`). Для создания этих списков откроем файл `hello/firstapp/views.py` и изменим в нем функцию `index`, как показано в листинге 6.50.

#### Листинг 6.50. Фрагмент файла `views.py`

```
def index(request):
    my_kv = ['I квартал ->', 'II квартал ->', 'III квартал->',
              'IV квартал->']
    my_month = ['Январь', 'Февраль', 'Март',
                'Апрель', 'Май', 'Июнь',
                'Июль', 'Август', 'Сентябрь',
                'Октябрь', 'Ноябрь', 'Декабрь']
    context = {'my_month': my_month, 'my_kv': my_kv}
    return render(request, "firstapp/index.html", context)
```

В этом программном коде созданы два списка: `my_kv` — названия кварталов, `my_month` — названия месяцев. Затем создан словарь `context`, в который включили два этих списка. В последней строке в функции `render` словарь `context` передан в шаблон `firstapp/index.html`.

Все изменения сделаны. Если обратиться к странице `contact.html` после этих изменений, то мы получим следующий результат (рис. 6.60).

Как видно из данного рисунка, список из 12 месяцев представлен на странице в виде таблицы из 4 столбцов и 4 строк, хотя в программном коде отсутствуют теги для фор-

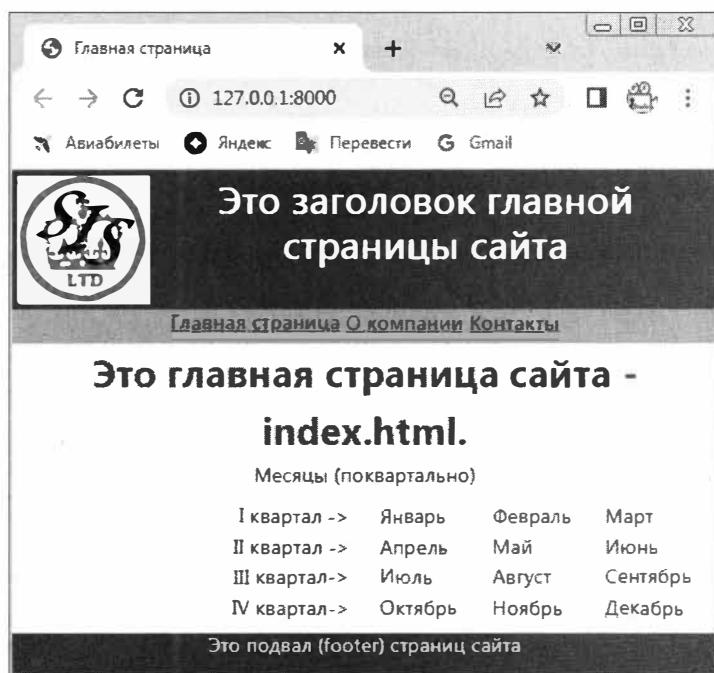


Рис. 6.60. Страница сайта `index.html` (использование Bootstrap для макетирования шаблона страницы)

мирования таблиц. Вся разбивка списка месяцев на столбцы и строки выполнена с использованием тегов `<div>` и классов фреймворка Bootstrap, а также за счет тегов организации циклов `{% for %}` фреймворка Django.

## 6.15. Краткие итоги

В этой главе были приведены элементарные теоретические сведения о шаблонах Django и показано, как можно создать шаблоны и передать в шаблоны различные данные из программы на Python. Мы узнали, что через шаблоны можно выдать пользователю ответы на его запросы, а сделать внешний вид шаблонов привлекательным можно с использованием классов фреймворка Bootstrap. Однако динамичные веб-сайты должны выполнять и обратную функцию, а именно — принять от пользователя некоторые данные и сохранить их в БД. Для этого в Django применяются *формы*. Так что пора перейти к следующей главе, в которой рассматриваются возможности взаимодействия пользователей с удаленной базой данных сайта через формы Django.





## ГЛАВА 7

# Формы

Форма HTML — это группа из одного или нескольких полей (виджетов) на веб-странице, которая служит для получения информации от пользователей для последующей отправки на сервер и представляет собой гибкий механизм сбора пользовательских данных. Формы несут целый набор виджетов для ввода различных типов данных: текстовые поля, флажки, переключатели, установщики дат и пр. Формы являются относительно безопасным способом взаимодействия пользовательского клиента и сервера, поскольку позволяют отправлять данные в POST-запросах, применяя защиту от межсайтовой подделки запроса (Cross Site Request Forgery, CSRF). Из материалов этой главы вы узнаете:

- что такое формы Django и для чего они нужны;
- как применять в формах POST-запросы;
- какие поля можно использовать в формах;
- как можно выполнить настройку формы и ее полей;
- как изменить внешний вид формы;
- как осуществляется проверка (валидация) данных в формах;
- как можно добавить стили к полям формы.

### 7.1. Процесс управления формами в Django

Изучение начнем с краткого обзора традиционных форм HTML. Рассмотрим простую форму HTML, имеющую одно поле для ввода, например, имени клиента (рис. 7.1).

Эта форма будет описана на языке HTML как набор элементов, расположенных внутри парных тегов `<form>...</form>`.

The screenshot shows a simple HTML form with a single input field and an 'OK' button. The input field is labeled 'Имя клиента:' and contains the placeholder text 'Введите ФИО'. The 'OK' button is located to the right of the input field.

Рис. 7.1. Поле формы для ввода ФИО клиента

```
<form action="/name_url/" method="post">
    <label for="name">Имя клиента: </label>
    <input id="name" type="text" name="name_field" value="Введите ФИО">
    <input type="submit" value="OK">
</form>
```

Здесь есть только одно поле для ввода ФИО, но форма может иметь любое количество элементов ввода и связанных с ними текстовых меток. Атрибут элемента `type` определяет, какой тип виджета будет показан в данной строке. Атрибуты `name` и `id` позволяют однозначно идентифицировать данное поле в JavaScript/CSS/HTML, в то время как `value` содержит значение для поля (когда оно показывается в первый раз). Текстовая метка добавляется при помощи тега `label` ("Имя клиента:") и имеет атрибут `for` со значением идентификатора `id` того поля, с которым данная текстовая метка связана.

Любая форма содержит как минимум одно поле-тег `input` типа `type="submit"`. Элемент `input` с `type="submit"` будет показан на форме как кнопка (по умолчанию), нажав на которую пользователь отправляет введенные им данные на сервер. Атрибуты формы определяют, каким методом будут отправлены данные на сервер (атрибут `method`) и куда (атрибут `action`):

- `action` — это ресурс (URL-адрес), куда будут отправлены данные для обработки. Если значение не указано (значение поля пустая строка), то данные будут отправлены в отображение (функцию, или класс), которое сформировало текущую страницу;
- `method` — это HTTP-метод, вызываемый для отправки данных: POST или GET.

Метод POST должен использоваться тогда, когда отправка данных приведет к внесению изменений в базе данных на сервере. Данный метод позволяет повысить уровень защиты от угроз CSRF.

Метод GET должен применяться только для форм, действия с которыми не приводят к изменению базы данных (например, для поисковых запросов). Кроме того, с помощью данного метода рекомендуется создавать внешние ссылки на ресурсы сайта.

Роль сервера заключается в отрисовке начального состояния формы, которая будет содержать либо пустые поля, либо поля с установленными начальными значениями. После того как пользователь нажмет на кнопку отправки, сервер получит все данные формы, а затем должен выполнить их проверку (валидацию). В том случае, если форма содержит неверные данные, сервер должен снова отрисовать форму, показав при этом поля с правильными данными, а также сообщения, описывающие ошибки пользователя. В тот момент, когда сервер получит запрос с "правильными" данными, он должен выполнить все необходимые действия (например, сохранение данных в БД, возврат результата поиска, загрузка файла и т. п.), а затем в случае необходимости проинформировать пользователя о выполненных действиях.

Как видно из этого описания, работа по проектированию форм может быть достаточно сложной. Разработчикам нужно описать структуру формы на языке HTML, проверить ее валидность, проверять введенные пользователем данные (на стороне клиента и на стороне сервера). При возникновении ошибок необходимо опять показать пользователю форму и указать на ошибки, в случае же успеха проделать с данными необходимые операции и каким-то образом проинформировать об этом пользователя.

При работе с формами Django берет большую часть описанных действий на себя, позволяя в программном коде определить ее поля как объекты, а затем использовать эти

объекты для генерации кода HTML-формы для контроля над процессом валидации и для других пользовательских взаимодействий с формой. Форма Django выполняет три наиболее важные части работы, связанных с данными:

- получение, реструктуризация данных и их подготовка к рендерингу (к визуализации);
- создание HTML-форм для ввода данных;
- получение и обработка данных от клиента и их запись в БД.

Весь этот код можно написать и вручную, но Django все это может сделать за вас. Последовательность действий Django при работе с формами представлена на рис. 7.2.

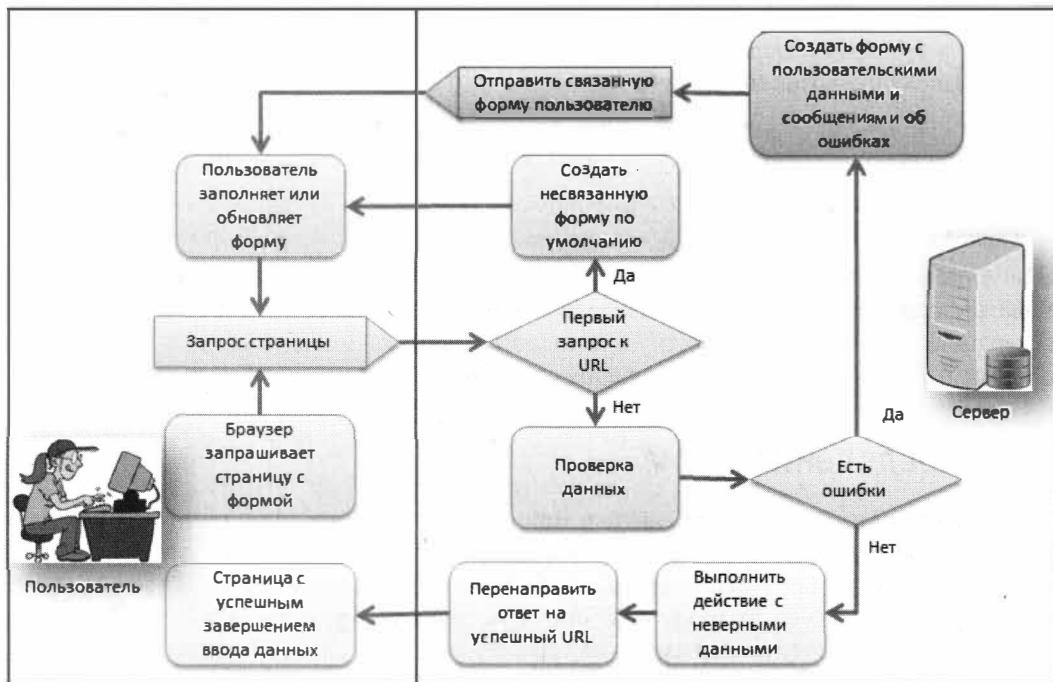


Рис. 7.2. Последовательность действий Django при работе с формами

В соответствии с рис. 7.2 Django выполняет следующие операции при работе с формами.

1. Показ формы по умолчанию при первом запросе со стороны пользователя.
  - Форма может содержать пустые поля (например, если вы создаете новую запись в базе данных), или поля могут иметь начальные значения (например, если вы изменяете запись или хотите заполнить ее каким-либо начальным значением).
  - Форма в текущий момент является *несвязанной*, потому что она не ассоциируется с какими-либо данными, введенными пользователем (хотя и может иметь начальные значения).
2. Получение данных из формы (из HTML-формы) со стороны клиента и связывание их с формой (классом формы) на стороне сервера.

- Связывание данных с формой означает, что данные, введенные пользователем, а также возможные ошибки при перерисовке в дальнейшем будут относиться именно к этой форме, а не к какой-либо еще.

### 3. Очистка и валидация данных.

- Очистка данных — это их проверка на наличие возможных значений или вставок в поля ввода (например, удаление неправильных символов, которые потенциально могут привести к отправке вредоносного содержимого на сервер) с последующей конвертацией очищенных данных в подходящие типы данных Python.
  - Валидация проверяет значения полей (например, правильность введенных дат, их диапазон и т. д.)
4. Если какие-либо данные ошибочны, то осуществляется перерисовка формы, но на этот раз с уже введенными пользователем данными и сообщениями об ошибках, описывающих возникшие проблемы.
5. Если все данные верны, то выполняются необходимые действия (например, сохранение данных, отправка писем, возврат результата поиска, загрузка файла и т. д.).
6. Когда все действия успешно завершены, пользователь перенаправляется на одну из страниц сайта.

Django предоставляет несколько инструментов, которые помогают создавать и обрабатывать формы. Наиболее фундаментальный из них — класс `Form`, который упрощает генерацию HTML-формы, очистку и валидацию данных. В следующих разделах описан процесс работы с формами на основе простых примеров.

## 7.2. Определение форм

Если вы планируете создавать сайты и приложения, которые принимают и сохраняют данные от пользователей, вам потребуются формы. Django предоставляет широкий набор инструментов для этого. Формы в HTML позволяют пользователю вводить текст, выбирать опции, изменять различные объекты, загружать рисунки на страницы и т. п., а потом отправлять эту информацию на сервер.

Django обеспечивает различные возможности по работе с формами. Можно определить функциональность форм в одном месте и затем использовать их многократно в разных местах. При этом упрощается проверка корректности данных, связывание форм с моделями данных и многое другое.

В Django основой для создания собственных форм является класс `Form`, который описывает форму и определяет, как она работает и отображается. Для того чтобы применять формы в своих проектах, нужно импортировать модуль `forms`, задав следующую инструкцию:

```
from django import forms
```

После этого можно создавать собственные классы форм на основе следующей инструкции:

```
class NameForm(forms.Form)
```

где `NameForm` — это имя класса вашей формы.

Как видно из данной инструкции, каждая форма определяется в виде отдельного класса, который расширяет класс `forms.Form`. Классы размещаются внутри проекта, где они используются. Как правило, они помещаются в отдельный файл, который можно назвать, например, `forms.py`. Однако формы могут размещаться и внутри других, уже имеющихся в приложении файлов, например в `views.py` или `models.py`. Чтобы не запутаться и четко знать, где находятся классы, описывающие формы, создадим для них отдельный файл.

Для работы с формами воспользуемся сайтом, который был создан в предыдущей главе. В проекте `hello` в папке с приложением `firstapp` создадим новый файл `hello/firstapp/forms.py`. В этом файле создадим нашу первую форму с именем `UserForm`. Для этого поместим в файл `forms.py` код из листинга 7.1, как показано на рис. 7.3.

### Листинг 7.1. Файл forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField()
    age = forms.IntegerField()
```

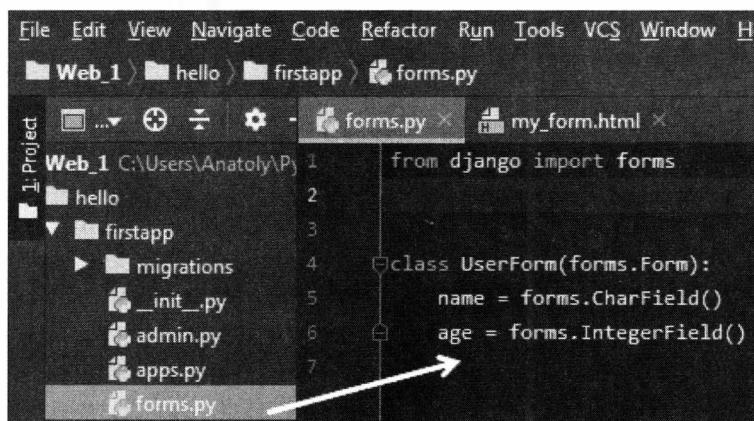


Рис. 7.3. Добавление файла `forms.py` в приложение `firstapp`

Здесь класс нашей новой формы имеет имя `UserForm`. В этом классе определены два поля: `name` (имя) имеет тип `forms.CharField` — текстовое поле (`input type="text"`), поле `age` (возраст) имеет тип `forms.IntegerField` — числовое поле (`input type="number"`). Таким образом, первое поле предназначено для ввода текста, а второе — для ввода чисел.

Далее в файле `views.py` изменим код функции `index()` и напишем код для новой функции `my_form()`. В итоге файл `views.py` будет выглядеть так, как показано в листинге 7.2 (изменения выделены серым фоном).

### Листинг 7.2. Изменения в файле views.py

```
from django.shortcuts import render
from .forms import UserForm
```

```

def index(request):
    my_text = 'Изучаем формы Django!'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
def my_form(request):
    my_form = UserForm()
    context = {"form": my_form}
    return render(request, "firstapp/my_form.html", context)

```

Здесь первым шагом был выполнен импорт созданного нами класса `UserForm`. Затем был изменен код функции `index()`, которая вызывает главную форму приложения. Здесь убраны строки формирования списков кварталов и месяцев, которые были написаны в последнем разделе главы 6, вместо них вставлена строка, которая будет отправлена в шаблон главной формы приложения. Наконец, была сформирована новая функция с именем `my_form`. В ней на основе класса `UserForm()` создан объект — `userform`, фактически это и есть наша форма из файла `forms.py`. Через словарь `context` эта форма передана в шаблон Django с именем `my_form.html`. Здесь созданный нами объект формы передается в шаблон `my_form.html` через ключ словаря `"form"`.

Создадим для нашей формы новый шаблон Django, это будет HTML-файл с именем `hello/templates/firstapp/my_form.html`. Код этого файла приведен в листинге 7.3.

### Листинг 7.3. Файл `my_form.html`

```

{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}

{% endblock content %}

```

Это дочерний шаблон или расширение базового шаблона с именем `firstapp/base.html`. В нем с помощью тега Bootstrap создан контейнер `class="container-fluid"`, в котором находится таблица HTML. В тело этой таблицы вставлен тег Django `{{ form }}`, т. е. та форма, которая была передана в шаблон из функции `my_form`.

Поскольку были внесены изменения в функцию `index()`, из которой делается обращение к главной странице сайта, нужно изменить код шаблона этой страницы. Изменим код шаблона страницы `hello/templates/firstapp/index.html` так, как показано в листинге 7.4 (изменения выделены серым фоном).

**Листинг 7.4. Изменения в файле index.html**

```
{% extends "firstapp/base.html" %}  
{% block title %}Главная страница{% endblock title %}  
{% block header %}Это главная страница сайта - index.html.{% endblock header %}  
{% block content%}  
    <div class="container-fluid my-2 text-start">  
        <div class="row">  
            <div class="col text-center text-dark">  
                <div>{{ my_text }}</div>  
            </div>  
        </div>  
    </div>  
    {% endblock content %}
```

На главной странице сайта теперь будет выведена всего одна строка, переданная из функции `index()` в словаре `context` по ключу `my_text`.

Теперь изменим базовый шаблон `hello/templates/firstapp/base.html` — добавим в главное меню сайта ссылку, по которой можно вызвать нашу форму. Изменения, внесенные в базовый шаблон `base.html`, приведены в листинге 7.5 (изменения выделены серым фоном).

**Листинг 7.5. Изменения в файле base.html**

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    {% load static %}  
    <meta charset="UTF-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1">  
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >  
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>  
    <title>{{ block title }}{{ endblock title }}</title>  
</head>  
<body>  
    <div class="container-fluid p-1 bg-primary text-white text-center">  
        <div class="row">  
            <div class="col-2 ">  
                  
                <h1>Это заголовок главной страницы сайта</h1>  
            </div>  
        </div>  
    </div>  
    <div class="container-fluid">  
        <div class="row bg-warning text-center">  
            <h6>  
                <a href="{% url 'index' %}">Главная страница</a>  
                <a href="{% url 'my_form' %}">Форма</a>  
                <a href="{% url 'about' %}">О компании</a>  
                <a href="{% url 'contact' %}">Контакты</a>  
            </h6>
```

```

        </h6>
    </div>
</div>
<div class="container-fluid">
    <div class="row text-center text-primary fs-1 fw-bold">
        <div>{% block header%}{% endblock header %}</div>
    </div>
    <div class="row text-center text-body">
        <div>{% block content%}{% endblock content %}</div>
    </div>
</div>
<div class="container-fluid">
    <div class="row bg-primary text-center text-white lh-1">
        <p>Это подвал (footer) страниц сайта</p>
    </div>
</div>
</body>
</html>

```

По сути изменения были внесены всего в одну строку:

```
<a href="{% url 'my_form' %}">Форма</a>
```

Здесь в главное меню добавлена ссылка на адрес формы с именем 'my\_form'.

Так как все ссылки на шаблоны страниц сайта находятся в файле `hello/firstapp/urls.py`, то добавим в данный файл строку со ссылкой на нашу форму (листинг 7.6, изменения выделены серым фоном).

#### Листинг 7.6. Изменения в файле urls.py

```

from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
]

```

Теперь, когда пользователь в главном меню сайта выберет опцию «Форма», то по связке `{% url 'my_form' %}` будет вызвана функция `views.my_form`, которая отобразит шаблон `my_form.html`.

Итак, были сделаны все шаги для того, чтобы «прописать» форму на нашем сайте. Если после этих изменений запустить локальный веб-сервер

```
python manage.py runserver
```

и перейти на главную страницу `http://127.0.0.1:8000/`, то мы увидим следующий результат (рис. 7.4).

Как видно из данного рисунка, в строке меню появилась ссылка, по которой можно вызвать форму Django. Если активировать эту ссылку, то будет выполнен переход на шаблон страницы сайта с созданной нами формой (рис. 7.5).

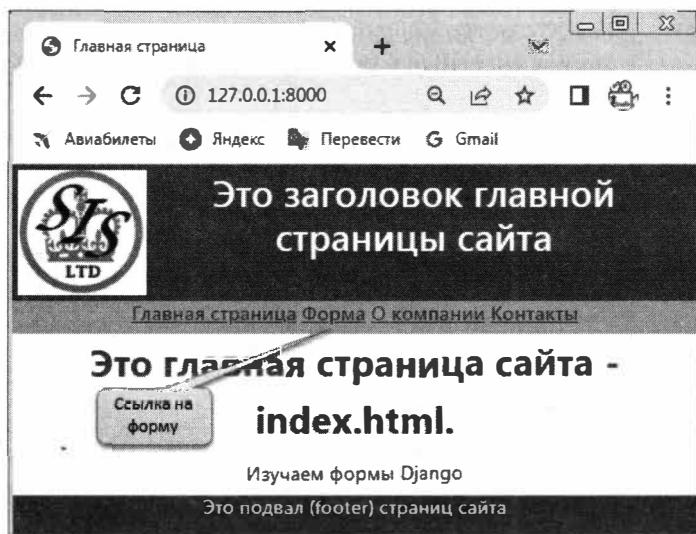


Рис. 7.4. Главная страница сайта со ссылкой на форму в меню

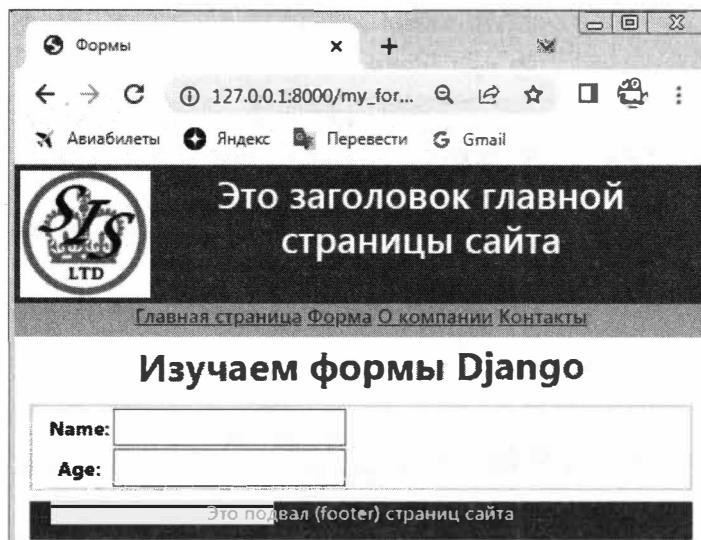


Рис. 7.5. Шаблон страницы сайта my\_form..html в окне браузера

На этой странице сайта в поля, которые обрамлены рамками, пользователь может вводить свои данные, причем в поле **Age** можно ввести только цифровые данные, ввод любого текстового символа будет заблокирован.

Следует заметить, что в модуле `forms.py` мы задали только идентификаторы полей `name` и `age` и не указали метки полей (`label`), которые должны выводиться на экран рядом с этими полями. В Django эти метки генерируются автоматически. В частности, как можно видеть, для наших двух полей были сгенерированы метки: **Name** и **Age**. По умолчанию в Django в качестве метки берется имя поля, и его первый символ меняется на заглавную букву. С одной стороны, такой прием избавляет программиста от дополн-

нительного задания значения метки поля, но это не всегда удобно. Часто требуется задавать более осмысленные и достаточно длинные значения для меток полей и на языке, отличном от английского.

В Django имеется возможность задания метки для поля с помощью параметра `label`. Немного изменим содержимое файла `forms.py` и добавим к полям метки, как показано в листинге 7.7.

#### Листинг 7.7. Изменения в файле forms.py

```
from django import forms

class UserForm(forms.Form):
    name = forms.CharField(label= "Имя клиента")
    age = forms.IntegerField(label="Возраст клиента")
```

После этих изменений шаблон страницы `my_form.html` будет выглядеть как на рис. 7.6.

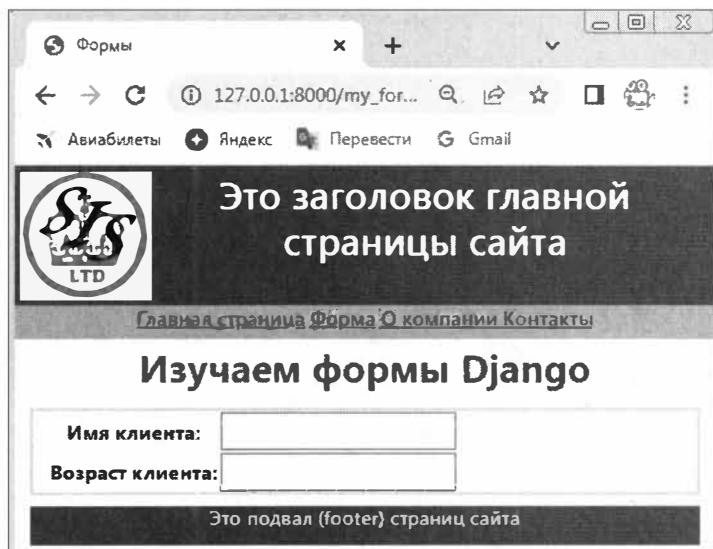


Рис. 7.6. Вид формы `forms.py` в окне браузера при явном определении меток полей

Кроме параметра `label` поля форм Django имеют еще ряд параметров, которые разработчик может определять программным способом. На них мы остановимся более подробно в следующих разделах.

## 7.3. Использование полей в формах Django

### 7.3.1. Настройка среды для изучения полей разных типов

В формах Django используются собственные классы для создания полей, через которые пользователь может вводить в приложение различные признаки и данные. Для демон-

страгии примеров применения различных полей мы воспользуемся файлом-шаблоном для форм `firstapp\my_form.html`. В этот файл внесем небольшие изменения (листинг 7.8), которые выделены серым фоном.

#### Листинг 7.8. Изменения в файле `my_form.html`

```
{% extends "firstapp/base.html" %}  
{% block title %}Формы{% endblock title %}  
{% block header %}Изучаем формы Django{% endblock header %}  
{% block content %}  
<div class="container-fluid text-center my-2 border border-3">  
    <form method="POST">  
        {% csrf_token %}  
        <table>  
            {{ form }}  
        </table>  
        <input type="submit" value="Отправить" >  
    </form>  
</div>  
{% endblock content %}
```

Здесь был добавлен стандартный элемент HTML `<form>` с указанием метода отправки данных на сервер, которые пользователь введет в поля формы. Различают два метода отправки данных — GET и POST:

- GET — отправляет данные на сервер через адресную строку (URL-адрес);
- POST — отправляет данные на сервер в теле HTML-запроса.

Так как данные в форме могут содержать конфиденциальную информацию, то в таких случаях нужно использовать метод POST.

Затем в теле тега формы `<form>` помещен встроенный тег Django `{% csrf_token %}`, который позволяет защитить приложение от CSRF-атак, добавляя в форму csrf-токен в виде скрытого поля.

#### ПРИМЕЧАНИЕ

CSRF (Cross-Site Request Forgery, также XSRF) — опаснейшая атака, которая приводит к тому, что хакер может выполнить на неподготовленном сайте массу различных действий от имени других зарегистрированных посетителей.

В нижней части формы помещена кнопка для отправки содержимого этой формы на сервер:

```
<input type="submit" value="Отправить" >
```

Данные, которые сервер получает из формы, служат для идентификации пользователей, либо записываются в БД, либо являются параметрами для поиска информации в БД. В примерах данной главы мы не будем реально отправлять данные на сервер, эти функции будут рассмотрены в следующих главах, но эта кнопка будет задействована для имитации отправки данных.

Все изучаемые нами поля мы будем размещать в коде файла `forms.py`.

### 7.3.2. Типы полей в формах Django и их общие параметры

Для начала мы познакомимся со списком наиболее употребляемых полей, которые есть в формах Django (табл. 7.1), и рядом свойств, общих для всех типов полей. А после этого более детально разберем примеры использования полей.

**Таблица 7.1. Список основных полей, используемых в формах Django**

№ п/п	Тип поля	Описание типа поля
1	BooleanField	Создает поле checkbox
2	CharField	Предназначено для ввода текста
3	ChoiceField (choises=кортеж_кортежей)	Генерирует список select, каждый из его элементов формируется на основе отдельного кортежа. Например, =(1, "English"), (2, "German"), (3, "French")
4	DateTimeField()	Устанавливает дату. В создаваемое поле вводится текст, который может быть сконвертирован в дату, например: 2021-12-25 или 11/25/21
5	DecimalField()	Ввод чисел с дробной частью
6	DurationField()	Предназначено для ввода временного промежутка. Вводимый текст должен соответствовать формату "DD HH:MM:SS", например 2 1:10:20 (2 дня 1 час 10 минут 20 секунд)
8	EmailField	Ввод адреса электронной почты
9	FileField()	Предназначено для выбора файла
10	FilePathField (path="каталог файлов")	Создает список select, который содержит все папки и файлы в определенном каталоге
11	FloatField()	Ввод чисел с плавающей точкой
12	GenericIPAddressField	Ввод IP-адреса
13	ImageField()	Предназначено для выбора файла, но при этом добавляет ряд дополнительных возможностей
14	IntegerField()	Ввод целых чисел
15	JSONField	Поле, которое принимает данные в формате JSON
16	MultipleChoiceField (choises=кортеж_кортежей)	Также генерирует список select на основе кортежа, как и forms.ChoiceField, добавляя к создаваемому полю атрибут multiple="multiple". Список поддерживает множественный выбор
17	NullBooleanField	Создает поле выбора (не выбрано, да, нет)
18	RegexField (regex="рег_выр")	Для ввода текста, который должен соответствовать определенному регулярному выражению

Таблица 7.1 (окончание)

№ п/п	Тип поля	Описание типа поля
19	SlugField()	Для ввода текста, который условно называется «slug» — это последовательность символов в нижнем регистре, чисел, дефисов и знаков подчеркивания
20	TimeField()	Для ввода времени — например: 14:30:59 или 14:30
21	TypeChoiceField (choises=кортеж_кортежей, coerce=функция_преобразования, empty_value=None)	Также генерирует список select на основе кортежа. Однако дополнительно принимает функцию преобразования, которая преобразует каждый элемент. И также принимает параметр empty_value, который указывает на значение по умолчанию
22	TypedMultipleChoiceField (choises=кортеж_кортежей, coerce=функция_преобразования, empty_value=None)	Аналог forms.TypeChoiceField для списка с множественным выбором
23	URLField()	Ввод ссылок
24	UUIDField()	Ввод UUID (универсального уникального идентификатора)
Сложные встроенные классы		
25	ComboField (fields=[field1, field2,...])	Аналогично обычному текстовому полю, однако требует, чтобы вводимый текст соответствовал условиям тех полей, которые передаются через параметр fields
26	MultiValueField (fields=[field1, field2,...])	Предназначено для создания сложных компоновок, состоящих из нескольких полей
27	forms.SplitDateTimeField()	Создает два текстовых поля для ввода соответственно даты и времени

Каждое поле, когда оно отображается на HTML-странице, имеет типичный для него внешний вид. Например, поле для ввода текста имеет обрамление в виде рамки. За внешний вид полей отвечает *виджет* — визуальный элемент на HTML-странице. Виджеты не следует путать с полями формы. Поля формы имеют дело с логикой проверки ввода и используются непосредственно в шаблонах. Виджеты же имеют дело с отображением (внешним видом) полей формы на веб-странице и извлечением необработанных отправляемых данных. Однако виджеты должны быть назначены полям формы.

Всякий раз, когда вы создаете поле в форме, Django задает виджет по умолчанию, соответствующий типу данных. Ранее рассмотренным полям при генерации разметки соответствовали определенные виджеты из пакета forms.widgets. Например, класс CharField — виджет forms.widgets.TextInput, а класс ChoiceField — виджет forms.widgets.Select. Однако есть ряд виджетов, которые по умолчанию не связаны с полями форм, но тем не менее мы можем их применять:

- PasswordInput — генерирует поле для ввода пароля: <input type="password" >;
- HiddenInput — генерирует скрытое поле: <input type="hidden" >;
- MultipleHiddenInput — генерирует набор скрытых полей;
- TextArea — генерирует многострочное текстовое поле: <textarea></textarea>;
- RadioSelect — генерирует список переключателей (радиокнопок): <input type="radio" >;
- CheckboxSelectMultiple — генерирует список флажков: <input type="checkbox">;
- TimeInput — генерирует поле для ввода времени (например, 12:41 или 12:41:32);
- SelectDateWidget — генерирует три поля select для выбора дня, месяца и года;
- SplitHiddenDateTimeWidget — использует скрытое поле для хранения даты и времени;
- FileInput — генерирует поле для выбора файла.

Каждое поле, которое размещено на форме, ориентируется на собственную логику проверки введенных данных, а также принимает несколько других аргументов, которые можно назначить при инициализации поля.

Каждый экземпляр класса Field имеет метод clean(), который принимает единственный аргумент и вызывает исключение django.forms.ValidationError в случае ошибки или возвращает чистое значение:

Field.clean(value)

Некоторые классы Field принимают дополнительные аргументы. Приведенные далее аргументы принимаются всеми полями:

- required — указывает на необходимость обязательного заполнения поля (по умолчанию: required=True). Чтобы сделать поле необязательным для заполнения, нужно указать: required=False;
- label — определяет видимую пользователем метку для поля (например, если имя поля name и на экране нужно показать «**Ваше имя**», то это можно сделать следующим образом: name = forms.CharField(label='Ваше имя');
- label\_suffix — переопределяет атрибут формы label\_suffix для каждого поля. Например:

```
num = forms.IntegerField(label='Номер', label_suffix='->')
```

- initial — определяет начальное значение для поля при его отображении на незаполненной форме. Например:

```
name = forms.CharField(initial='Введите ФИО')
url = forms.URLField(initial='http://')
```

- widget — позволяет указать класс Widget, который следует использовать при отображении поля вместо того, который задан по умолчанию;
- help\_text — описание данных, которые нужно внести в поле. Если вы укажете help\_text, он будет показан около поля при отображении формы с помощью вспомогательных методов. Например:

```
subject = forms.CharField(max_length=100, help_text='Не более 100 символов')
sender = forms.EmailField(help_text='Введите email адрес')
```

- `error_messages` — изменяет стандартные сообщения об ошибках, которые выдает поле. Создайте словарь с ключами тех сообщений, которые вы желаете изменить. Если не задать значение этому аргументу:

```
generic = forms.CharField()
```

то будет выдано стандартное сообщение об ошибке: **This field is required.**

Если этот аргумент задать явно, например:

```
name = forms.CharField(error_messages={'required':  
    'Ошибка, не введено имя'})
```

то будет выдано заданное сообщение об ошибке: **Ошибка, не введено имя;**

- `validators` — позволяет указать список функций, осуществляющих проверку поля на корректность введенных данных;
- `localize` — включает локализацию для данных формы, как на входе, так и на выходе;
- `disabled` — этот аргумент принимает булево значение. При значении `True` — запрещает редактировать значение поля, которое задано по умолчанию. Даже если пользователь отправит данные этого поля, они будут проигнорированы и принято значение по умолчанию.

Итак, приступим к изучению полей, приведенных в табл. 7.1, на простых примерах. Для всех примеров будем менять код в одном и том же файле `hello/firstapp/forms.py`, в котором создан класс `UserForm` для формирования пользовательских форм.

### 7.3.3. Поле **BooleanField** для выбора решения: да/нет

Метод `forms.BooleanField` создает поле `<input type="checkbox">` (флажок). Это поле возвращает значение типа `Boolean`: `True` — если флажок отмечен и `False` — если флажок не отмечен.

При инициализации этого поля по умолчанию устанавливается виджет `CheckboxInput` и возвращаемое значение `False`, при этом флажок в `checkbox` отсутствует. Пример создания такого поля приведен в листинге 7.9.

#### Листинг 7.9. Изменения в файле `forms.py`

```
from django import forms  
class UserForm(forms.Form):  
    basket = forms.BooleanField(label="Положить товар в корзину")
```

Если мы внесем этот код в модуль `hello/firstapp/forms.py` и запустим наше приложение, то получим следующее отображение данного поля на форме (рис. 7.7).

Если клиент не захочет положить товар в корзину, оставив это поле пустым, и нажмет кнопку **Отправить**, то в ответ получит следующее сообщение (рис. 7.8).

Похоже, что логика работы приложения нарушена: клиент не хочет приобретать некий товар, но приложение вынуждает его установить флажок в поле выбора. Это происходит потому, что для всех полей по умолчанию установлен признак обязательности вне-

сения значений в поле (`required=True`), а при инициализации поля оно не заполнено (в нем нет флашка). Чтобы пользователь имел возможность оставить это поле пустым, нужно при инициализации поля отключить требование обязательности его заполнения, т. е. свойству `required` присвоить значение `false`. Иными словами, при создании в приложении этого поля потребуется следующий код (листинг 7.10).

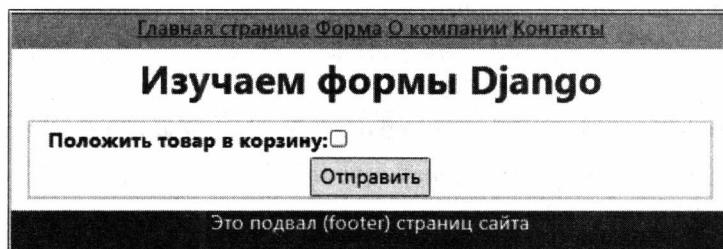


Рис. 7.7. Поле BooleanField на странице my\_form.html

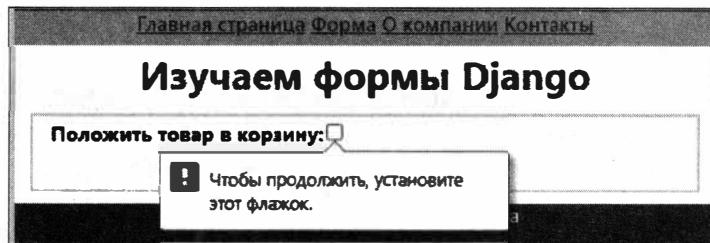


Рис. 7.8. Ответ из формы после нажатия кнопки Отправить на странице my\_form.html

#### Листинг 7.10. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    basket = forms.BooleanField(label="Положить товар в корзину",
                                required=False)
```

В этом случае пользователь может либо поставить флашок в поле `basket`, либо оставить его пустым, и приложение будет работать корректно.

#### 7.3.4. Поле `CharField` для ввода текста

Метод `forms.CharField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

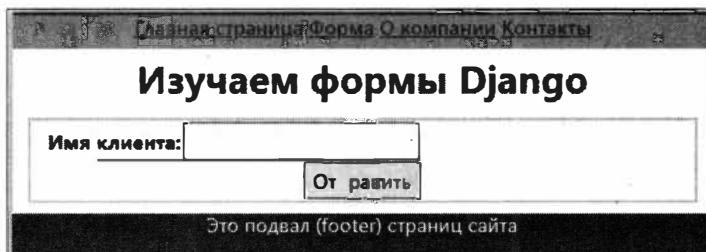
Это поле служит для ввода текста. По умолчанию здесь используется виджет `TextInput`, начальное значение которого — пустая строка (значение `None`) или текст, который был задан в свойстве поля `empty_value`.

Изменим модуль `forms.py` и создадим там это поле с помощью кода листинга 7.11.

**Листинг 7.11. Изменения в файле forms.py**

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.9).



**Рис. 7.9.** Поле CharField на странице my\_form.html

Не забываем, что в таком варианте по умолчанию это поле будет обязательным для заполнения. Если пользователь оставит его пустым, то при нажатии на кнопку **Отправить** ему будет выдано напоминание, и программа не даст продвинуться вперед до тех пор, пока поле не будет заполнено. Если логикой программы допускается оставить это поле пустым, то его нужно создавать с помощью следующего программного кода:

```
name = forms.CharField(label="Имя клиента", required=False)
```

Число символов, которые пользователь может ввести в поле CharField, можно задать с помощью следующих аргументов:

- max\_length — максимальное число символов в поле;
- min\_length — минимальное число символов в поле.

Например, мы можем создать это поле с помощью такого кода:

```
name = forms.CharField(label="Имя клиента", max_length=15,
                      help_text="ФИО не более 15 символов")
```

В этом случае пользователь не сможет внести в поле более чем 15 символов, и рядом с полем появится дополнительная подсказка. Внесем изменение в модуль forms.py (листинг 7.12).

**Листинг 7.12. Изменения в файле forms.py**

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента", max_length=15,
                          help_text="ФИО не более 15 символов")
```

Если теперь загрузить форму в окно браузера, то получим следующий результат (рис. 7.10).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Имя клиента:

ФИО не более 15 символов

Это подвал (footer) страниц сайта

Рис. 7.10. Подсказка об ограничении числа символов в поле CharField

### 7.3.5. Поле *ChoiceField* для выбора данных из списка

Метод forms.ChoiceField создает на HTML-странице следующую разметку:

```
<select>
    <option value="1">Date 1</option>
    <option value="2"> Date 2</option>
    <option value="3"> Date 3</option>
</select>
```

Поле ChoiceField служит для выбора данных из списка и по умолчанию использует виджет Select с пустым значением None.

Поле имеет структуру forms.ChoiceField(choices=кортеж кортежей) и генерирует список select, каждый из элементов которого формируется на основе отдельного кортежа. Например, поле

```
forms.ChoiceField(choices=((1, "Английский"),
                           (2, "Немецкий"),
                           (3, "Французский")))
```

сформирует список для выбора из трех элементов.

Поле ChoiceField принимает один аргумент — choices. Это либо итерация из двух кортежей, используемая в качестве элемента выбора для этого поля, либо вызываемая функция, которая возвращает такую итерацию.

Изменим модуль forms.py и создадим там такое поле с помощью кода листинга 7.13.

#### Листинг 7.13. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    ling = forms.ChoiceField(label="Выберите язык",
                            choices=((1, "Английский"),
                                    (2, "Немецкий"),
                                    (3, "Французский")))
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.11).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Выберите язык: Английский

Отправить

Это подвал (footer) страниц сайта

Рис. 7.11. Поле ChoiceField на странице my\_form.html

Если пользователь нажмет левую кнопку мыши в этом поле, то откроется список для выбора одного из языков (рис. 7.12).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Выберите язык: Английский

Английский  
Немецкий  
Французский

Отправить

Это подвал (footer) страниц сайта

Рис. 7.12. Раскрывающийся список для выбора данных в поле ChoiceField

### 7.3.6. Поле *DateField* для ввода даты

Метод `forms.DateField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Поле `DateField` служит для ввода дат (например, `2021-12-25` или `25/12/2021`) и по умолчанию использует виджет `DateInput` с пустым значением `None`. При этом делается проверка, является ли введенное значение либо строкой `datetime.date`, либо датой в определенном формате. Поле принимает один необязательный аргумент `input_formats`.

Изменим модуль `forms.py` и создадим там поле для ввода даты с помощью кода листинга 7.14.

#### Листинг 7.14. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    date = forms.DateField(label="Введите дату")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.13).

Рис. 7.13. Поле DateField на странице my\_form.html

### 7.3.7. Поле *DateTimeField* для ввода даты и времени

Метод `forms.DateTimeField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Поле `DateTimeField` служит для ввода даты и времени (например, `2021-12-25 14:30:59` или `25/12/2021 14:30`) и по умолчанию использует виджет `DateTimeInput` с пустым значением `None`. При этом делается проверка, является ли введенное значение либо строкой `datetime.datetime`, `datetime.date`, либо значениями даты и времени в определенном формате. Поле имеет один необязательный аргумент `input_formats`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.15.

#### Листинг 7.15. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    date_time = forms.DateTimeField(label="Введите дату и время")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.14).

Рис. 7.14. Поле DateTimeField на странице my\_form.html

### 7.3.8. Поле *DecimalField* для ввода десятичных чисел

Метод `forms.DecimalField` создает на HTML-странице следующую разметку:

```
<input type="number">
```

Поле `DecimalField` служит для ввода десятичных чисел и по умолчанию использует виджет `NumberInput` с пустым значением `None`. При этом делается проверка, является ли вводимое число десятичным.

Поле принимает пять необязательных аргументов:

- `max_value` — максимальное значение вводимого числа;
- `min_value` — минимальное значение вводимого числа;
- `max_whole_digits` — максимальное число цифр (до десятичной запятой плюс после десятичной запятой, с разделенными начальными нулями), допустимое в значении;
- `decimal_places` — максимально допустимое число знаков после запятой;
- `step_size` — шаг изменения введенного значения поля с использованием стрелок прокрутки.

Сообщения об ошибках, выдаваемые при выходе вводимого значения за пределы `max_value` и `min_value`, могут содержать тег `%{limit_value}s`, замещаемый соответствующим пределом. Аналогично сообщения об ошибках при выходе вводимых значений за пределы, определяемые параметрами `max_digits`, `max_decimal_places` и `max_whole_digits`, могут содержать тег `%{max}s`.

Когда курсор мыши находится в данном поле, то в правой его части появляются стрелки прокрутки. Нажимая на верхнюю или нижнюю стрелку левой кнопкой мыши, можно увеличивать или уменьшать значение, введенное в данное поле. С помощью аргумента `step_size` (размер шага) можно указать число, на которое будет меняться значение введенного числа.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.16.

#### Листинг 7.16. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    num = forms.DecimalField(label="Введите десятичное число")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.15).

В поле `DecimalField` невозможен ввод никаких символов, кроме чисел и разделителя дробной части, а также значений, выходящих за пределы заданных необязательных



Рис. 7.15. Поле `DecimalField` на странице `my_form.html`

аргументов. Чтобы показать это, изменим программный код и укажем, что дробная часть числа ограничена двумя знаками (листинг 7.17).

#### Листинг 7.17. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    num = forms.DecimalField(label="Введите десятичное число",
                            decimal_places=2)
```

Если теперь в поле `DecimalField` попытаться ввести число с более длинной дробной частью, то будет выдано предупреждение об ошибке ввода (рис. 7.16).

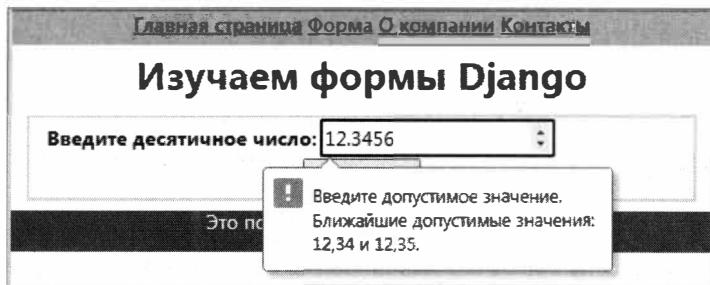


Рис. 7.16. Предупреждение об ошибке при вводе в поле `DecimalField` числа с дробной частью, превышающей заданное ограничение

### 7.3.9. Поле `DurationField` для ввода промежутка времени

Метод `forms.DurationField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Поле `DurationField` предназначено для ввода временного промежутка. Вводимый текст должен соответствовать формату "DD HH:MM:SS" — например, 2 1:10:20 (2 дня 1 час 10 минут 20 секунд). По умолчанию поле использует виджет `TextInput` с пустым значением `None`. При этом делается проверка, является ли введенное значение строкой, которую можно преобразовать в `timedelta`. Поле принимает два аргумента для проверки длины вводимой строки: `datetime.timedelta.min` и `datetime.timedelta.max`. Значение должно быть заключено между этими величинами.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.18.

#### Листинг 7.18. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    time_delta = forms.DurationField
    (label="Введите промежуток времени")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.17).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Введите промежуток времени:

Отправить

Это подвал (footer) страниц сайта

Рис. 7.17. Поле DurationField на странице my\_form.html

### 7.3.10. Поле *EmailField* для ввода электронного адреса

Метод `forms.EmailField` создает на HTML-странице следующую разметку:

```
<input type="email">
```

Это поле служит для ввода электронного адреса.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.19.

#### Листинг 7.19. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    email = forms.EmailField(label="Электронный адрес",
                           help_text="Обязательный символ - @")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.18).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Электронный адрес:

Обязательный символ - @

Отправить

Это подвал (footer) страниц сайта

Рис. 7.18. Поле EmailField на странице my\_form.html

Поле `EmailField` по умолчанию использует виджет `EmailInput` с пустым значением `None` и метод `EmailValidator` с умеренно сложным регулярным выражением, проверяющим, что введенное в поле значение является действительным адресом электронной почты.

Поле принимает два необязательных аргумента для проверки числа введенных символов: `max_length` — максимальная длина строки и `min_length` — минимальная длина строки. Они гарантируют, что длина введенной строки не превышает или равна заданным в этих аргументах значениям.

Если пользователь ошибся и ввел некорректный электронный адрес, то при нажатии на кнопку **Отправить** ему будет выдано напоминание, и программа не даст продвинуться вперед до тех пор, пока поле не будет заполнено правильно (рис. 7.19).

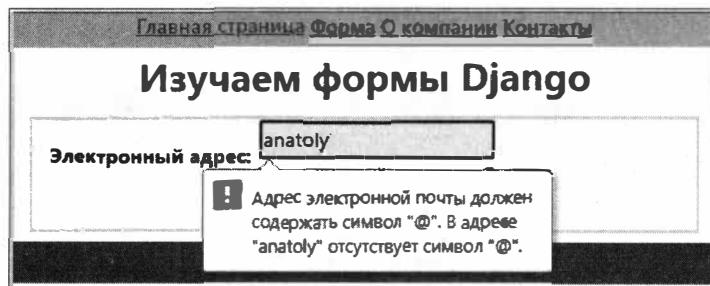


Рис. 7.19. Подсказка об ошибке при вводе электронного адреса в поле EmailField

### 7.3.11. Поле *FileField* для выбора файлов

Метод `forms.FileField` создает на HTML-странице следующую разметку:

```
<input type="file">
```

Поле `FileField` предназначено для выбора и загрузки файлов и по умолчанию использует виджет `ClearableFileInput` с пустым значением `None`. Поле формирует объект `UploadedFile`, который упаковывает содержимое файла и имя файла в один объект. Поле принимает два необязательных аргумента для проверки длины вводимой строки: `max_length` (гарантирует, что имя файла не превысит максимальную заданную длину) и `allow_empty_file` (гарантирует, что проверка пройдет успешно, даже если содержимое файла пустое).

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.20.

#### Листинг 7.20. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    file = forms.FileField(label="Файл")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.20).

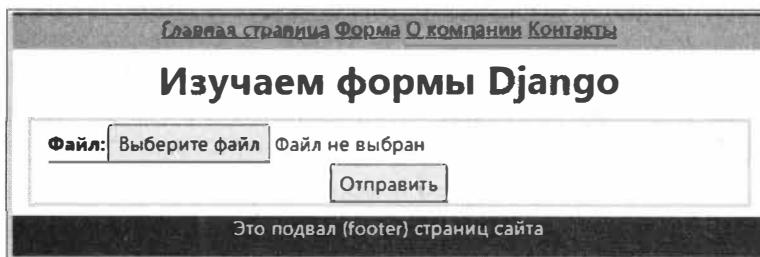


Рис. 7.20. Поле *FileField* на странице my\_form.html

Как можно видеть из данного рисунка, поле `FileField` представлено в виде кнопки с надписью «Выберите файл» и сообщением «Файл не выбран». Если теперь нажать на кнопку Выберите файл, то откроется новое окно, в котором можно перемещаться по любым папкам компьютера, просматривать и выбирать произвольные файлы (рис. 7.21).

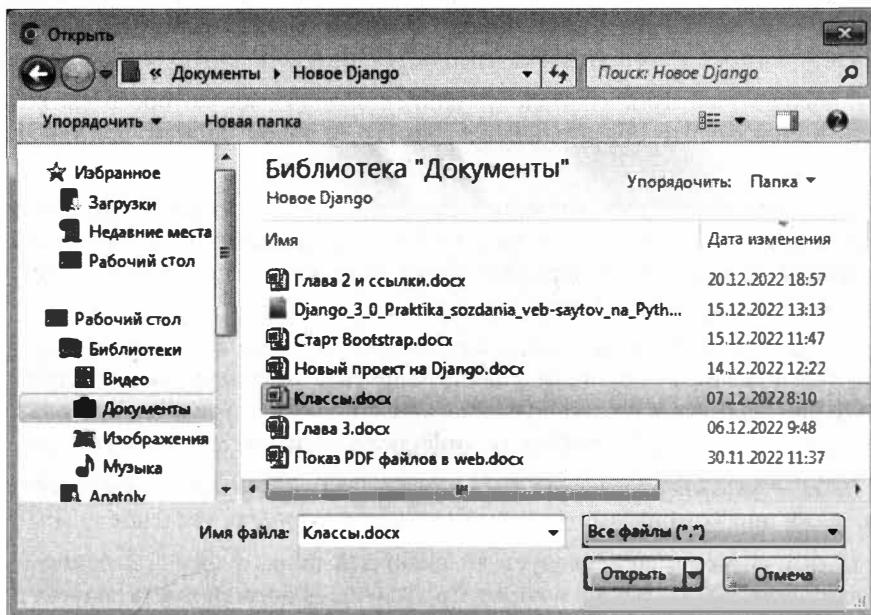


Рис. 7.21. Возможность выбора файла из окна ОС Windows при активации поля `FileField`

Предположим, что был выбран файл `Классы.docx` и нажата кнопка `Открыть`, — окно Windows будет закрыто, а имя выбранного файла показано на веб-странице (рис. 7.22).



Рис. 7.22. Поле `FileField` на странице `my_form.html` после того, как пользователь выбрал файл

### 7.3.12. Поле `FilePathField` для создания списка файлов

Метод `forms.FilePathField` создает на HTML-странице следующую разметку:

```
<select>
<option value="folder/file1">folder/file1</option>
```

```
<option value="folder/file2">folder/file2</option>
<option value="folder/file3">folder/file3</option>
//.....
</select>
```

Поле `FilePathField` по умолчанию использует виджет `Select` с пустым значением `None` и проверяет, существует ли выбранный вариант в списке вариантов. Ключи сообщений об ошибках: `required`, `invalid_choice`. Поле позволяет делать выбор из файлов внутри определенного каталога. При этом ему требуются пять дополнительных аргументов:

- `path` — абсолютный путь к каталогу (папке), содержимое которого вы хотите отобразить в списке. Это обязательный параметр, и такой каталог должен существовать;
- `recursive` — разрешает или запрещает доступ к подкаталогам. Если этот параметр имеет значение `False` (по умолчанию), то будут предложены к выбору файлы только в указанном каталоге. Если параметр имеет значение `True`, то к выбору будут предложены все вложенные каталоги;
- `match` — этот параметр представляет шаблон регулярного выражения (будут отображаться только файлы с именами, совпадающими с этим шаблоном). Регулярное выражение применяется к названию файла, а не к полному пути. Например, выражение `"foo.*\.txt$"` покажет файл `foo23.txt`, но отфильтрует файлы `bar.txt` или `foo23.gif`;
- `allow_files` — указывает, следует ли включать файлы в заданном каталоге (по умолчанию `True`), при этом параметр `allow_folders` должен иметь значение `True`;
- `allow_folders` — указывает, следует ли включать подкаталоги в заданном каталоге (по умолчанию `False`), при этом параметр `allow_files` должен иметь значение `True`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.21.

#### Листинг 7.21. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    file_path = forms.FilePathField(label="Выберите файл",
                                    path="C:/my_doc/")
```

Здесь мы задали возможность выбора файла из папки `C:\my_doc\`.

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.23).

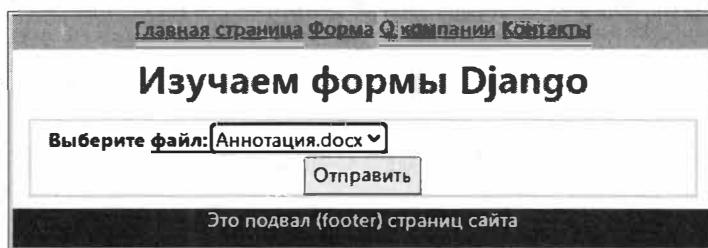


Рис. 7.23. Поле `FilePathField` на странице `my_form.html`

Если мы теперь щелкнем мышью на этом поле, то откроется список файлов, которые содержатся в указанном каталоге. Пользователь будет иметь возможность выбора любого файла из предложенного списка (рис. 7.24).

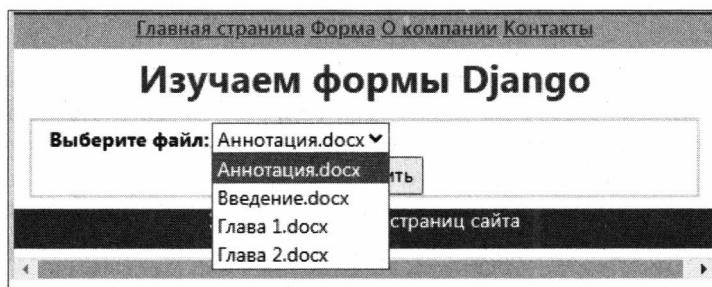


Рис. 7.24. В поле FilePathField предоставлена возможность выбора файла из списка доступных

Внесем еще одно изменение в модуль forms.py, с помощью которого реализуем возможность отображать в поле FilePathField не только файлы, но и вложенные каталоги (листинг 7.22).

#### Листинг 7.22. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    file_path = forms.FilePathField(label="Выберите файл",
                                    path="C:/my_doc/",
                                    allow_files="True",
                                    allow_folders="True")
```

Если теперь запустить наше приложение, то мы увидим, что кроме файлов у нас появилась возможность выбора и каталогов (рис. 7.25).

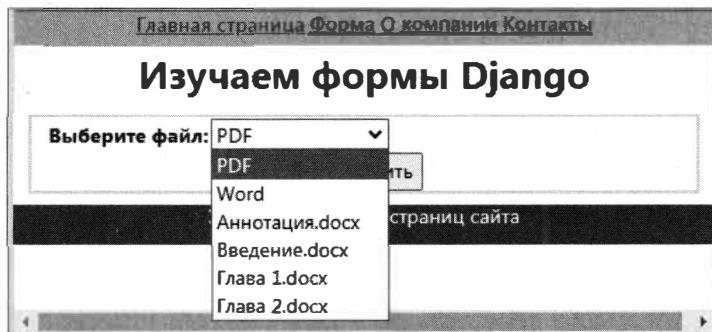


Рис. 7.25. В поле FilePathField предоставлена возможность выбора файла и вложенных каталогов из списка доступных

**7.3.13. Поле *FloatField***  
для ввода чисел с плавающей точкой

Метод `forms.FloatField` создает на HTML-странице следующую разметку:

```
<input type="number">
```

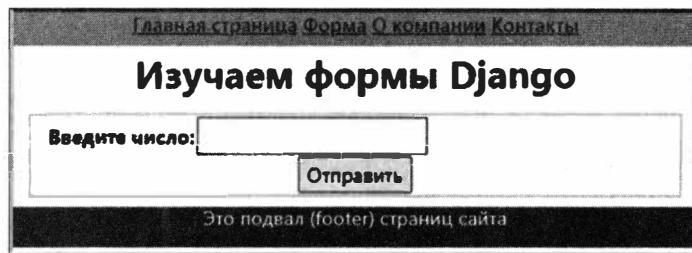
Поле `FloatField` служит для ввода чисел с плавающей точкой и по умолчанию использует виджет `NumberInput` с пустым значением `None`. При этом делается проверка, является ли вводимое число числом с плавающей точкой. Поле принимает два необязательных аргумента для проверки максимального (`max_value`) и минимального (`min_value`) значений вводимого числа, контролирующие диапазон значений, разрешенных в поле.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.23.

### Листинг 7.23. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    num = forms.FloatField(label="Введите число")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.26).



**Рис. 7.26.** Поле `FloatField` на странице `my_form.html`

### **7.3.14. Поле *GenericIPAddressField* для ввода IP-адреса**

Метод `forms.GenericIPAddressField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле служит для ввода IP-адреса.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.24.

#### Листинг 7.24. Изменения в файле forms.py

Главная страница Форма О компании Контакты

## Изучаем формы Django

IP адрес:  Пример формата 192.0.2.0

Это подвал (footer) страниц сайта

Рис. 7.27. Поле GenericIPAddressField на странице my\_form.html

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.27).

Поле GenericIPAddressField по умолчанию использует виджет TextInput с пустым значением None, при этом проверяется, что введенное значение является действительным IP-адресом.

### 7.3.15. Поле *ImageField* для выбора файлов изображений

Метод forms.ImageField создает на HTML-странице следующую разметку:

```
<input type="file">
```

Поле ImageField предназначено для выбора и загрузки файлов, представляющих собой изображения, и по умолчанию использует виджет ClearableFileInput с пустым значением None. Поле формирует объект UploadedFile, который упаковывает содержимое файла и имя файла в один объект.

Изменим модуль forms.py и создадим там такое поле с помощью кода листинга 7.25.

#### Листинг 7.25. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    file = forms.ImageField(label="Изображение")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.28).

Главная страница Форма О компании Контакты

## Изучаем формы Django

Изображение:  Выберите файл Файл не выбран

Это подвал (footer) страниц сайта

Рис. 7.28. Поле ImageField на странице my\_form.html

Как можно видеть, поле `ImageField` представлено кнопкой с надписью «Выберите файл» и сообщением «Файл не выбран». Если теперь нажать на кнопку Выберите файл, то откроется новое окно, в котором можно перемещаться по любым папкам компьютера, просматривать и выбирать произвольные файлы (рис. 7.29).

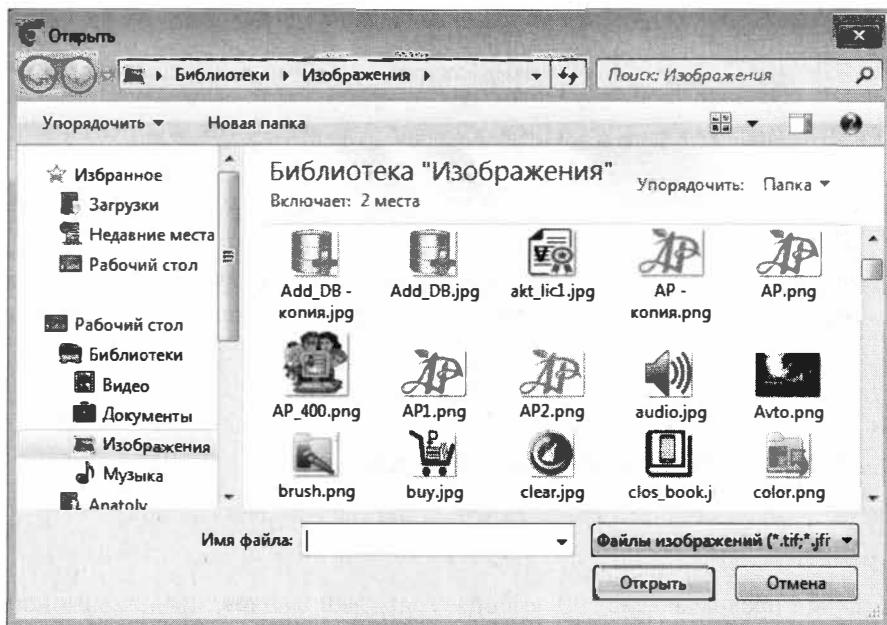


Рис. 7.29. Окно для выбора файлов с изображением

Предположим, что был выбран файл `AP_400.png` и нажата кнопка **Открыть**, — окно Windows будет закрыто, а имя выбранного файла показано на веб-странице (рис. 7.30).

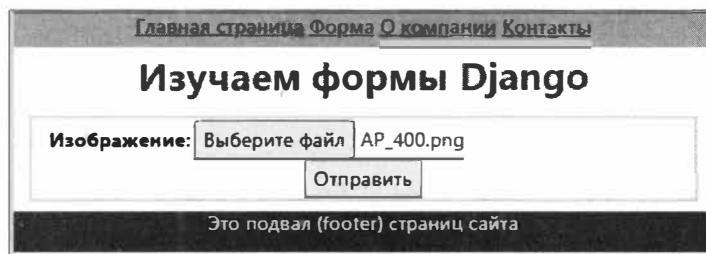


Рис. 7.30. Поле `ImageField` на странице `my_form.html` с выбранным файлом изображения

### 7.3.16. Поле `IntegerField` для ввода целых чисел

Метод `forms.IntegerField` создает на HTML-странице следующую разметку:

```
<input type="number">
```

Поле `IntegerField` служит для ввода целых чисел и по умолчанию использует виджет `NumberInput` с пустым значением `None`. При этом делается проверка, является ли вводимое

число целым. Поле принимает два необязательных аргумента для проверки максимального (`max_value`) и минимального (`min_value`) значений вводимого числа и использует методы `MaxValueValidator` и `MinValueValidator`, если эти ограничения заданы.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.26.

#### Листинг 7.26. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    num = forms.IntegerField(label="Введите целое число")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.31).

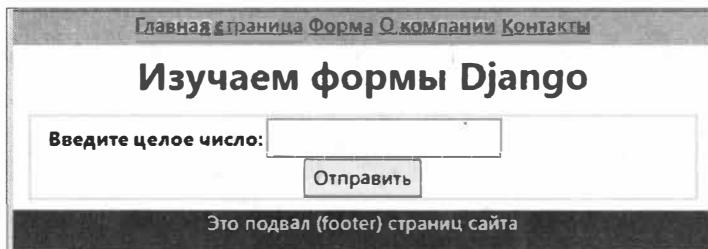


Рис. 7.31. Поле `IntegerField` на странице `my_form.html`

Если в поле `IntegerField` попытаться ввести число с дробной частью, то будет выдано предупреждение об ошибке ввода (рис. 7.32).

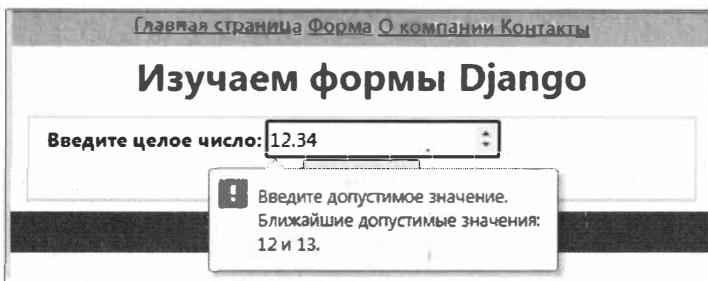


Рис. 7.32. Предупреждение об ошибке при вводе в поле `IntegerField` числа с дробной частью

### 7.3.17. Поле `JsonField` для данных формата JSON

В настоящее время на многих программных платформах часто встречаются данные формата JSON. Этот формат позволяет хранить частично структурированные данные вместе с другими полями данных в БД PostgreSQL, MySQL и SQLite. Гибкость полей JSON помогает легко извлекать и отображать данные, таким образом, эти поля в основном применяются на современном рынке ПО, и Django также использует их для форм и моделей.

Метод `forms.JSONField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Поле `JSONField` служит для ввода данных формата JSON и по умолчанию использует виджет `TextInput` с пустым значением `None`. При этом делается проверка, соответствуют ли вводимые данные формату JSON. Поле принимает два необязательных аргумента: `encoder` (для сериализации данных) и `decoder` (для десериализации данных).

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.27.

#### Листинг 7.27. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    num = forms.JSONField(label="Данные формата JSON")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.33).

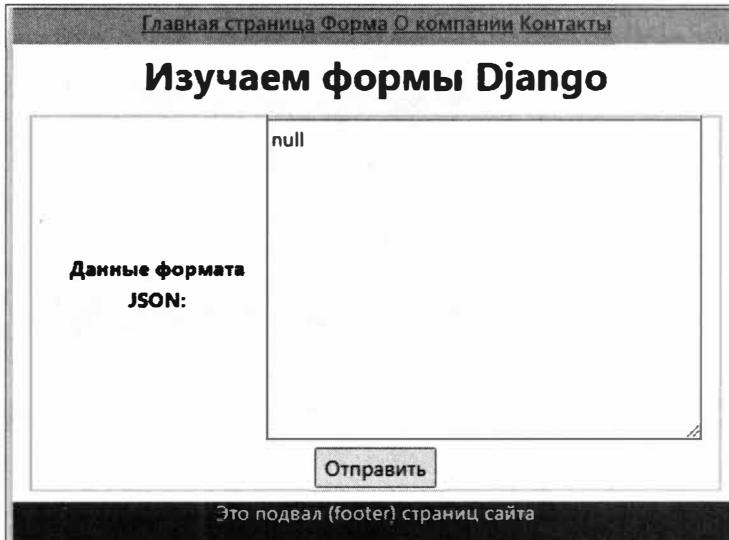


Рис. 7.33. Поле `JSONField` на странице `my_form.html`

Обратите внимание на правый нижний угол поля, там имеется специальная метка для изменения размера поля ввода. Если данную метку «зацепить» с помощью левой кнопки мыши, то можно уменьшать или увеличивать размер поля, что обеспечит более удобный ввод большого количества данных формата JSON.

Конечно, метод `JSONField` в большинстве случаев не очень удобен для рядовых пользователей, однако это полезный способ форматирования данных из виджета на стороне клиента для отправки на сервер, который вполне подходит для квалифицированных разработчиков.

### 7.3.18. Поле *MultipleChoiceField* для выбора данных из списка

Поле *MultipleChoiceField* имеет следующую структуру:

```
forms.MultipleChoiceField(choises=кортеж_кортежей)
```

Поле генерирует список select на основе кортежа, как и поле *forms.ChoiceField*, добавляя к создаваемому полю атрибут `multiple="multiple"` и обеспечивая тем самым поддержку множественного выбора.

По умолчанию поле *MultipleChoiceField* использует виджет *SelectMultiple* с пустым значением [] (пустой список), а также — как и поле *ChoiceField* — принимает один дополнительный обязательный аргумент *choices*.

Изменим модуль *forms.py* и создадим там такое поле с помощью кода листинга 7.28.

#### Листинг 7.28. Изменения в файле *forms.py*

```
from django import forms
class UserForm(forms.Form):
    country = forms.MultipleChoiceField(label="Выберите страны",
                                         choices=((1, "Англия"),
                                                   (2, "Германия"),
                                                   (3, "Испания"),
                                                   (4, "Россия")))
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.34).

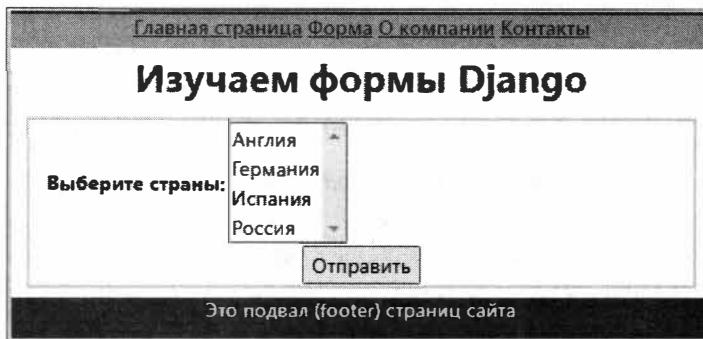


Рис. 7.34. Поле *MultipleChoiceField* на странице *my\_form.html*

Как можно видеть из данного рисунка, в поле одновременно выводится весь список, и пользователь имеет возможность выбрать из него несколько элементов. Для этого необходимо поочередно щелкнуть мышью на нужных элементах списка, удерживая при этом нажатой клавишу <Ctrl>, — выбранные пользователем элементы будут выделены цветом (рис. 7.35).

Как видно из данного рисунка, в рассматриваемом случае выбраны две страны: **Германия** и **Россия**.

Главная страница Форма О компании Контакты

## Изучаем формы Django

Выберите страны:

Англия  
Германия  
Испания  
Россия

Отправить

Это подвал (footer) страниц сайта

**Рис. 7.35.** В поле `MultipleChoiceField` предоставлена возможность выбора из списка нескольких элементов

### 7.3.19. Поле `NullBooleanField` для выбора решения: да/нет

Метод `forms.NullBooleanField` создает на HTML-странице следующую разметку:

```
<select>
<option value="1" selected="selected">Неизвестно</option>
<option value="2">Да</option>
<option value="3">Нет</option>
</select>
```

Изменим модуль `forms.py` и создадим там это поле с помощью кода листинга 7.29.

#### Листинг 7.29. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    vyb = forms.NullBooleanField(label="Вы поедете в Сочи этим летом?")
```

Если запустить наше приложение, то мы получим такое отображение этого поля на форме (рис. 7.36).

Поле `NullBooleanField` по умолчанию использует виджет `NullBooleanSelect` и имеет пустое значение `None`. В зависимости от действия пользователя оно может вернуть три значе-

Главная страница Форма О компании Контакты

## Изучаем формы Django

Вы поедете в Сочи этим летом?

Неизвестно

Отправить

Это подвал (footer) страниц сайта

**Рис. 7.36.** Поле `NullBooleanField` на странице `my_form.html`

ния: None (Неизвестно), True (Да) или False (Нет). При этом оно никогда и ничего не проверяет (т. е. не вызывает метод `ValidationError`). Если пользователь щелкнет левой кнопкой мыши в этом поле, то ему будут предложены три варианта выбора (рис. 7.37).

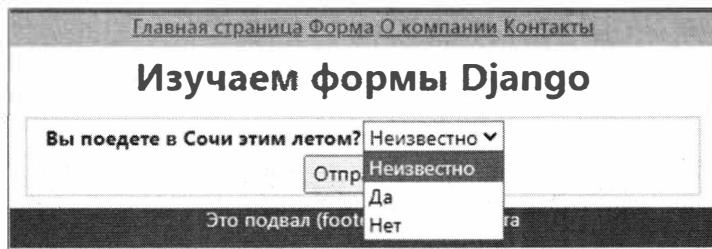


Рис. 7.37. Варианты выбора в поле `NullBooleanField` на странице `my_form.html`

### 7.3.20. Поле `RegexField` для ввода текста

Метод `forms.RegexField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле предназначено для ввода текста, который должен соответствовать определенному регулярному выражению.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.30.

#### Листинг 7.30. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    reg_text = forms.RegexField(label="Текст", regex="^[\dA-F][\dA-F]$")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.38).

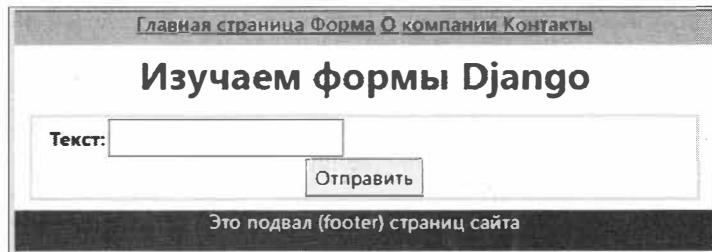


Рис. 7.38. Поле `RegexField` на странице `my_form.html`

Поле `RegexField` по умолчанию использует виджет `TextInput` с пустым значением `None` и метод `RegexValidator`, проверяющий, что введенное в поле значение соответствует определенному регулярному выражению, которое указывается в виде строки или скомпилированного объекта регулярного выражения.

Поле также принимает параметры: `max_length` — максимальная длина строки и `min_length` — минимальная длина строки, которые работают так же, как и для поля `CharField`.

### 7.3.21. Поле `SlugField` для ввода текста

Метод `forms.SlugField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле предназначено для ввода текста, который условно называется «`slug`» и представляет собой последовательность символов в нижнем регистре, чисел, дефисов и знаков подчеркивания.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.31.

#### Листинг 7.31. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    slug_text = forms.SlugField(label="Введите текст")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.39).

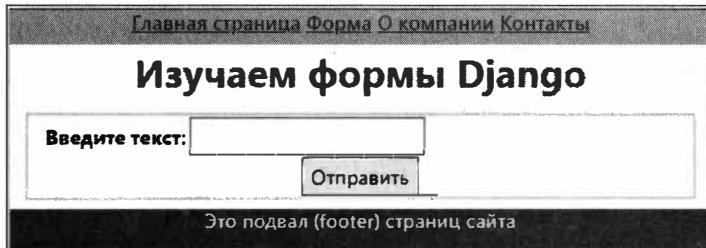


Рис. 7.39. Поле `SlugField` на странице `my_form.html`

Поле `SlugField` по умолчанию использует виджет `TextInput` с пустым значением `None` и методы `validate_slug` или `validate_unicode_slug`, проверяющие, что введенное значение содержит только буквы, цифры, подчеркивания и дефисы.

У поля есть также необязательный параметр `allow_unicode` — логическое указание для поля принимать символы Unicode в дополнение к символам ASCII. Значение по умолчанию — `False`.

### 7.3.22. Поле `TimeField` для ввода времени

Метод `forms.TimeField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Поле `TimeField` служит для ввода значений времени (например, 14:30:59 или 14:30) и по умолчанию использует виджет `TextInput` с пустым значением `None`. При этом выполняет-

ся проверка, является ли введенное значение либо строкой `datetime.time`, либо форматированным в определенном формате значением времени. Поле принимает один необязательный аргумент `input_formats`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.32.

#### Листинг 7.32. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    time = forms.DateTimeField(label="Введите время")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.40).

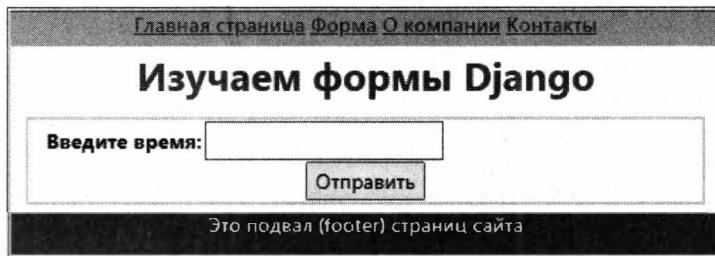


Рис. 7.40. Поле `TimeField` на странице `my_form.html`

### 7.3.23. Поле `TypedChoiceField` для выбора данных из списка

Метод `forms.TypedChoiceField` создает на HTML-странице следующую разметку:

```
<select>
    <option value="1">Date 1</option>
    <option value="2"> Date 2</option>
    <option value="3"> Date 3</option>
</select>
```

Поле `TypedChoiceField` служит для выбора данных из списка. Оно аналогично полю `ChoiceField` и по умолчанию также использует виджет `Select` с пустым значением `None`.

Поле имеет следующую структуру:

```
forms.TypedChoiceField(choises=кортеж_кортежей,
                      coerce=функция_преобразования,
                      empty_value=None)
```

Поле генерирует список `select` на основе кортежа, однако дополнительно принимает еще два аргумента:

- `coerce` — функцию преобразования, которая принимает один аргумент и возвращает его приведенное значение;

- `empty_value` — значение, используемое для представления «пусто». По умолчанию задана пустая строка.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.33.

#### Листинг 7.33. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    city = forms.TypedChoiceField(label="Выберите город",
                                   empty_value=None,
                                   choices=((1, "Москва"),
                                            (2, "Воронеж"),
                                            (3, "Курск")))
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.41).

Если пользователь щелкнет мышью в этом поле, то откроется список выбора (рис. 7.42).

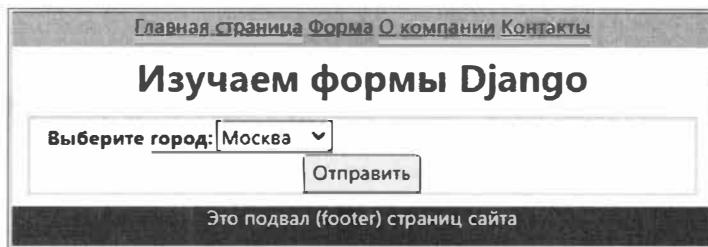


Рис. 7.41. Поле `TypedChoiceField` на странице `my_form.html`

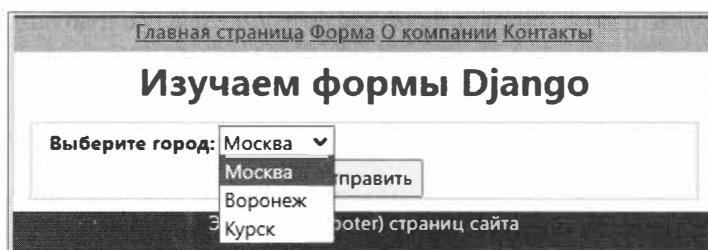


Рис. 7.42. Раскрывающийся список для выбора данных в поле `TypedChoiceField`

### 7.3.24. Поле `TypedMultipleChoiceField` для выбора данных из списка

Поле `TypedMultipleChoiceField` имеет следующую структуру:

```
forms.TypedMultipleChoiceField(choises=кортеж_кортежей,
                               coerce=функция_преобразования,
                               empty_value=None)
```

Поле генерирует список select на основе кортежа, как и поле forms.TypedChoiceField, добавляя к создаваемому полю атрибут multiple="multiple" и обеспечивая тем самым поддержку множественного выбора.

По умолчанию поле TypedMultipleChoiceField использует виджет SelectMultiple с пустым значением [] (пустой список), а также принимает два дополнительных аргумента:

- coerce — функцию преобразования, которая принимает один аргумент и возвращает его приведенное значение;
- empty\_value — значение, используемое для представления «пусто». По умолчанию задана пустая строка.

Изменим модуль forms.py и создадим там такое поле с помощью кода листинга 7.34.

#### Листинг 7.34. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    city = forms.TypedMultipleChoiceField(label="Выберите город",
                                           empty_value=None,
                                           choices=((1, "Москва"),
                                                     (2, "Воронеж"),
                                                     (3, "Курск"),
                                                     (4, "Томск")))
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.43).

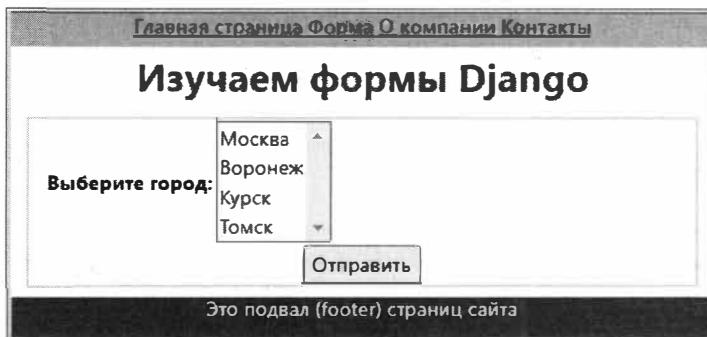


Рис. 7.43. Поле TypedMultipleChoiceField на странице my\_form.html

Как можно видеть, в поле одновременно выводится весь список, и пользователь имеет возможность выбрать из него несколько элементов. Для этого необходимо поочередно щелкнуть мышью на нужных элементах списка, удерживая при этом нажатой клавишу <Ctrl>, — выбранные пользователем элементы будут выделены цветом (рис. 7.44).

В рассматриваемом случае выбраны два города: **Воронеж** и **Томск**.

The screenshot shows a web page titled 'Изучаем формы Django'. A dropdown menu labeled 'Выберите город:' contains four options: 'Москва', 'Воронеж', 'Курск', and 'Томск'. The 'Томск' option is currently selected. Below the dropdown is a button labeled 'Отправить' (Send). At the bottom of the page, there is a footer bar with the text 'Это подвал (footer) страниц сайта'.

Рис. 7.44. В поле TypedMultipleChoiceField предоставлена возможность выбора нескольких элементов из списка

### 7.3.25. Поле *URLField* для ввода универсального указателя ресурса (URL)

Метод `forms.URLField` создает на HTML-странице следующую разметку:

```
<input type="url">
```

Это поле предназначено для ввода универсального указателя ресурса (URL), например, такого: `http://www.google.com`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.35.

#### Листинг 7.35. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    url_text = forms.URLField(label="Введите URL",
                               help_text="Например http://www.yandex.ru)
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.45).

Поле `URLField` по умолчанию использует виджет `URLInput` с пустым значением `None` и метод `URLValidator`, проверяющий, что введенное значение является действительным URL.

The screenshot shows a web page titled 'Изучаем формы Django'. A text input field is labeled 'Введите URL:' with the placeholder 'Например http://www.yandex.ru'. Below the input field is a button labeled 'Отправить' (Send). At the bottom of the page, there is a footer bar with the text 'Это подвал (footer) страниц сайта'.

Рис. 7.45. Поле `URLField` на странице `my_form.html`

Поле также принимает параметры: `max_length` — максимальная длина строки и `min_length` — минимальная длина строки, которые работают так же, как и для поля `CharField`. Если пользователь при вводе URL допустит ошибку в формате этого типа данных, ему будет выдано соответствующее предупреждение (рис. 7.46).

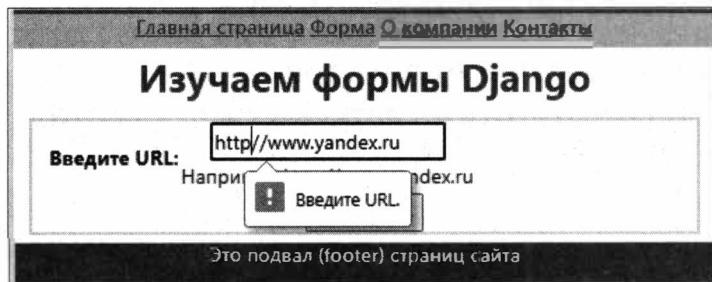


Рис. 7.46. Предупреждение об ошибке в поле `URLField`

### 7.3.26. Поле `UUIDField` для ввода универсального уникального идентификатора UUID

Метод `forms.UUIDField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле предназначено для ввода универсального уникального идентификатора UUID (например: 123e4567-e89b-12d3-a456-426655440000).

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.36.

#### Листинг 7.36. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    uuid_text = forms.UUIDField(label="Введите UUID",
                                help_text="Формат xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.47).

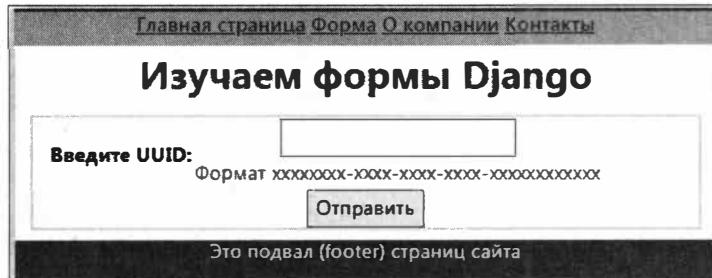


Рис. 7.47. Поле `UUIDField` на странице `my_form.html`

Поле `UUIDField` по умолчанию использует виджет `TextInput` с пустым значением `None`. Поле будет принимать любой формат строки, принятый в качестве HEX-аргумента для `UUID`-конструктора.

## 7.4. Встроенные классы для создания сложных полей

### 7.4.1. Поле `ComboField` для ввода текста с проверкой соответствия заданным форматам

Метод `forms.ComboField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле имеет структуру `forms.ComboField(fields=[field1, field2,...])` и один обязательный параметр `fields` — список полей, которые следует использовать для проверки значения поля (в том порядке, в котором они расположены при инициализации данного поля). По сути это аналог обычного текстового поля за тем исключением, что вводимый текст должен соответствовать требованиям тех полей, которые передаются через параметр `fields`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.37.

#### Листинг 7.37. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    combo_text = forms.ComboField(label='Введите данные',
                                   fields=[
                                       forms.CharField(max_length=20),
                                       forms.EmailField()])
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.48).

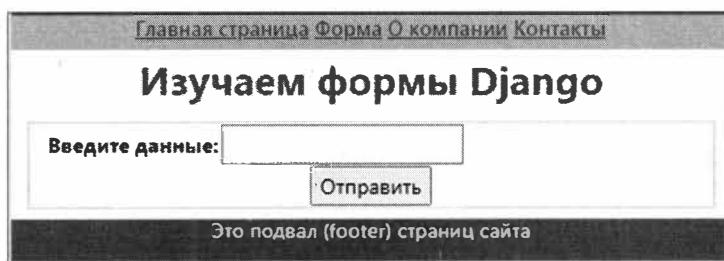


Рис. 7.48. Поле `ComboField` на странице `my_form.html`

Для поля `ComboField` выполняется проверка введенного значения на соответствие каждому из полей, указанных в качестве аргумента при его создании. Оно принимает один дополнительный обязательный аргумент: `fields`. В этом примере вводимые пользователем данные будут проверяться на соответствие формату поля для электронного адреса,

и чтобы число символов не превысило заданной величины (в нашем случае — 20 символов).

## 7.4.2. Поле *MultiValueField* для создания сложных компоновок из нескольких полей

Метод `forms.MultiValueField` создает на HTML-странице следующую разметку:

```
<input type="text">
```

Это поле агрегирует логику нескольких полей, которые вместе создают одно значение, и имеет один обязательный аргумент — `fields` — кортеж полей, значения которых очищаются и впоследствии объединяются в одно значение. Каждое значение поля очищается соответствующим полем в `fields`. Первое значение поля очищается первым полем, второе — вторым полем и т. д. После очистки всех полей список чистых значений объединяется в одно значение по `compress()`.

Это поле может принимать необязательные аргументы:

- `require_all_fields` — по умолчанию имеет значение `True`, в этом случае будет возникать ошибка проверки, если для какого-либо поля не указано значение;
- `widget` — значение по умолчанию — `TextInput`;
- `compress(список_данных)` — принимает список допустимых значений и возвращает «сжатую» версию этих значений в виде одного значения.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.38.

### Листинг 7.38. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    combo_text = forms.MultiValueField(label='Комплексное поле',
                                         fields=(forms.CharField(max_length=20),
                                                 forms.EmailField()))
```

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.49).

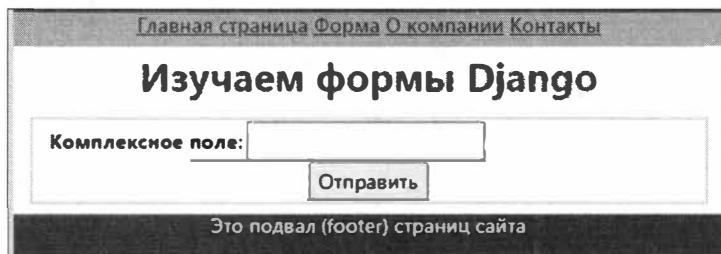


Рис. 7.49. Поле *MultiValueField* на странице `my_form.html`

Конечно, это очень упрощенный пример использования данного поля. Для того чтобы в полной мере задействовать возможности такого комплексного поля, нужно создавать на его основе самостоятельный класс, например:

```
class PhoneField(MultiValueField):
```

Поскольку для новичков применение поля этого типа не актуально, то в данной книге мы не будем приводить подробности.

### 7.4.3. Поле *SplitDateTimeField* для раздельного ввода даты и времени

Метод `forms.SplitDateTimeField` создает на HTML-странице следующую разметку:

```
<input type="text" name="_0" >
<input type="text" name="_1" >
```

Поле `SplitDateTimeField` предназначено для ввода даты и времени в два раздельных текстовых поля и по умолчанию использует виджет `SplitDateTimeWidget` с пустым значением `None`. В первое поле вводится дата, во второе поле — время. При этом делается проверка, является ли введенное значение либо строкой `datetime.datetime`, либо форматированными в определенном формате значениями даты и времени.

Поле принимает два необязательных аргумента:

- `input_date_formats` — список форматов, используемых для попытки преобразования строки в допустимый объект `datetime.date`. Если аргумент `input_date_formats` не указан, форматы ввода задаются по умолчанию для поля `DateField`;
- `input_time_formats` — список форматов, используемых для попытки преобразования строки в допустимый объект `datetime.time`. Если аргумент `input_time_formats` не указан, форматы ввода назначаются по умолчанию для поля `TimeField`.

Изменим модуль `forms.py` и создадим там такое поле с помощью кода листинга 7.39.

#### Листинг 7.39. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    date_time = forms.SplitDateTimeField
        (label="Введите дату и время")
```

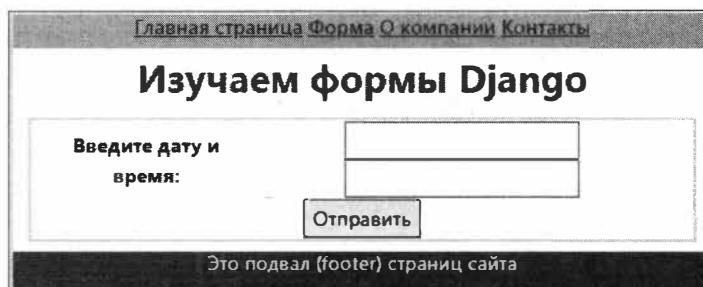


Рис. 7.50. Поле `SplitDateTimeField` на странице `my_form.html`

Если запустить наше приложение, то мы получим следующее отображение этого поля на форме (рис. 7.50).

## 7.5. Настройка формы и ее полей

В предыдущем разделе мы познакомились с различными типами полей для ввода данных и основными их свойствами — такими как, например, метка поля (`label`), подсказки (`help_text`) и т. п. Теперь познакомимся с тем, как можно менять некоторые другие свойства полей и их положение на веб-странице.

### 7.5.1. Изменение внешнего вида поля с помощью параметра `widget`

Параметр `widget` позволяет задать объект, который будет использоваться для генерации HTML-разметки и тем самым определять внешний вид поля на веб-странице. Если пользователь явно не указывает виджет, то Django применит тот виджет, который задан по умолчанию. Однако пользователь может указать для поля свой виджет, заменив им тот, который принят по умолчанию. Рассмотрим это на примере.

Изменим модуль `forms.py` и создадим там три поля с помощью кода листинга 7.40.

#### Листинг 7.40. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя")
    age = forms.IntegerField(label="Возраст")
    comment = forms.CharField(label="Комментарий")
```

Если запустить наше приложение, то мы получим следующее отображение этих полей на форме (рис. 7.51).

Поскольку мы явно не указали виджет для этих полей, то по умолчанию для них использовался виджет `TextInput`. Для ввода имени и возраста этот виджет вполне подходит, а вот для комментария он не совсем удобен, поскольку для ввода текста желательно более широкое поле. Назначим для поля ввода комментариев другой виджет — `Textarea`. Для этого изменим код в модуле `forms.py`, как показано в листинге 7.41.



Рис. 7.51. Три текстовых поля на странице `my_form.html` с виджетами `TextInput` по умолчанию

**Листинг 7.41. Изменения в файле forms.py**

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя")
    age = forms.IntegerField(label="Возраст")
    comment = forms.CharField(label="Комментарий",
                               widget=forms.Textarea)
```

Сделаем еще одно небольшое изменение в шаблоне формы, т. е. в файле `hello/templates/firstapp/my_form.html`.

Эти изменения выделены серым фоном в листинге 7.42.

**Листинг 7.42. Изменения в файле my\_form.html**

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}


<form method="POST">
    {% csrf_token %}
    <table>
        {{ form }}
    </table>
    <input type="submit" value="Отправить" >
</form>
</div>
{% endblock content %}


```



**Рис. 7.52.** Три текстовых поля на странице `my_form.html` с виджетом `Textarea` для поля с комментариями

Здесь мы изменили способ выравнивания текста в контейнере `container-fluid`, теперь он будет выровнен по левому краю.

Если после внесенных изменений запустить наше приложение, то мы получим следующее отображение этих полей на форме (рис. 7.52).

Как видно из данного рисунка, поле для ввода комментариев имеет больший размер, кроме того, в этом поле в правом нижнем углу появилась метка. Если данную метку «зашелпить» с помощью левой кнопки мыши, то можно уменьшать или увеличивать размер поля, что обеспечит более удобный ввод большого количества данных.

### 7.5.2. Задание начальных значений полей с помощью свойства *initial*

Когда в программе создается новое поле, то оно, как правило, имеет пустое значение. Однако его можно заполнить, воспользовавшись свойством `initial`.

Изменим модуль `forms.py`, задав полям для ввода имени и возраста начальные значения с помощью кода листинга 7.43.

#### Листинг 7.43. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя", initial="Введите ФИО")
    age = forms.IntegerField(label="Возраст", initial=18)
    comment = forms.CharField(label="Комментарий",
                               widget=forms.Textarea)
```

После этих изменений при запуске нашего приложения созданные поля не будут пустыми (рис. 7.53).

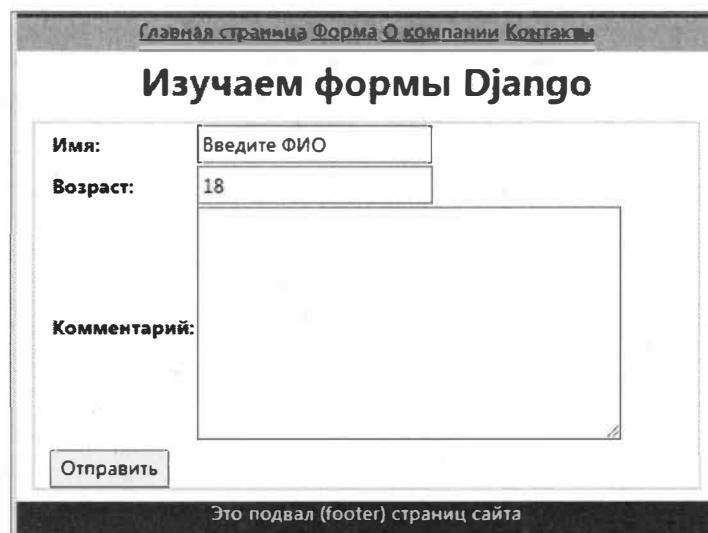


Рис. 7.53. Текстовые поля на странице `my_form.html` с заданными начальными значениями

### 7.5.3. Задание порядка следования полей на форме

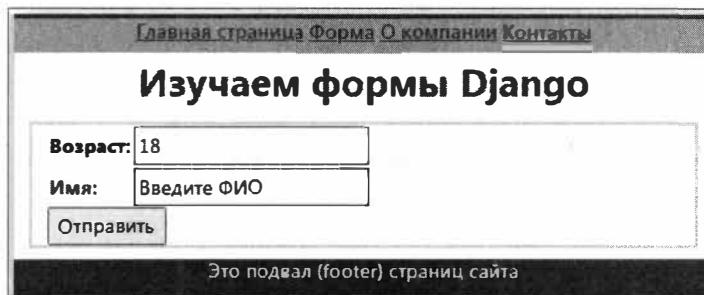
По умолчанию все поля на форме идут в той последовательности, в которой они описаны в модуле инициализации формы. Однако в процессе работы над проектом разработчик может многократно добавлять или удалять поля. Свойство `field_order` дает возможность гибко менять порядок следования полей на форме. Оно позволяет переопределить порядок следования полей как в классе формы, так и при определении объекта формы в представлении.

В классе формы это можно сделать, например, следующим образом (листинг 7.44).

#### Листинг 7.44. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя", initial="Введите ФИО")
    age = forms.IntegerField(label="Возраст", initial=18)
    field_order = ["age", "name"]
```

Здесь мы указали, что поле `age` будет отображаться на форме раньше, чем поле `name`, несмотря на то, что поле `name` было создано первым. При запуске приложения поля будут идти в той последовательности, как указано в свойстве `field_order` (рис. 7.54).



**Рис. 7.54.** Изменение порядка следования полей на форме с использованием свойства `field_order` в описании класса формы

Теперь изменим порядок следования полей при определении объекта формы в представлении. Для этого внесем изменение в программный код представления `views.py` — изменения выделены серым фоном (листинг 7.45).

#### Листинг 7.45. Изменения в файле views.py

```
from django.shortcuts import render
from .forms import UserForm
def index(request):
    my_text = 'Изучаем формы Django'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
```

```
def contact(request):
    return render(request, "firstapp/contact.html")
def my_form(request):
    my_form = UserForm(field_order=["age", "name"])
    context = {"form": my_form}
    return render(request, "firstapp/my_form.html", context)
```

А также изменим программный код в модуле `forms.py` (листинг 7.46).

#### Листинг 7.46. Изменения в файле `forms.py`

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя", initial="Введите ФИО")
    age = forms.IntegerField(label="Возраст", initial=18)
```

Здесь при описании класса формы поле `name` стоит на первом месте, а поле `age` — на втором. Однако в представлении `views.py` был задан обратный порядок следования этих полей с помощью свойства `field_order`:

```
userform = UserForm(field_order=["age", "name"])
```

В результате поля на форме будут выведены в такой последовательности — сначала `age`, а затем `name`.

### 7.5.4. Задание подсказок к полям формы

Чтобы предоставить пользователю дополнительную информацию о том, какие данные следует вводить в то или иное поле, можно использовать свойство поля `help_text`.

Восстановим естественный порядок следования полей при определении объекта формы в представлении. Для этого изменим программный код представления `views.py` (листинг 7.47).

#### Листинг 7.47. Изменения в файле `views.py`

```
from django.shortcuts import render
from .forms import UserForm
def index(request):
    my_text = 'Изучаем формы Django'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
def my_form(request):
    my_form = UserForm()
    context = {"form": my_form}
```

А в программный код в модуле `forms.py` добавим подсказки пользователю с использованием свойства `help_text` (листинг 7.48).

**Листинг 7.48. Изменения в файле forms.py**

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя", help_text ="Введите ФИО")
    age = forms.IntegerField(label="Возраст", help_text ="Введите возраст")
```

После этих изменений при запуске нашего приложения рядом с соответствующими полями появятся подсказки (рис. 7.55).

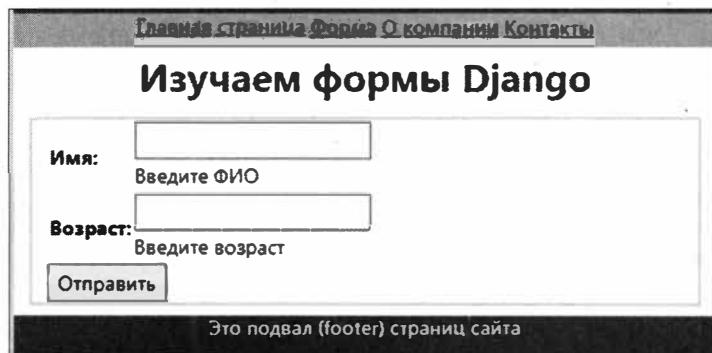


Рис. 7.55. Подсказки для полей формы с использованием свойства `help_text`

### 7.5.5. Настройки вида формы

Общее отображение полей на форме можно настроить с помощью специальных методов:

- `as_table()` — отображение полей в виде таблицы;
- `as_ul()` — отображение полей в виде списка;
- `as_p()` — отображение каждого поля формы в отдельном параграфе (абзаце).

Посмотрим, как работают эти методы на примере, для чего изменим шаблон нашей формы `firstapp\my_form.html` (листинг 7.49).

**Листинг 7.49. Изменения в файле my\_form.html**

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}


<h5>Форма как таблица</h5>
<form method="POST">
    {% csrf_token %}
    <table>
        {{ form.as_table  }}
    </table>


```

```
<input type="submit" value="Отправить" >
</form>
</div>
<div class="container-fluid text-start my-2 border border-3">
    <h5>Форма как список</h5>
    <form method="POST">
        {% csrf_token %}
        <ul>
            {{ form.as_ul }}
        </ul>
        <input type="submit" value="Отправить" >
    </form>
</div>
<div class="container-fluid text-start my-2 border border-3">
    <h5>Форма как параграф</h5>
    <form method="POST">
        {% csrf_token %}
        <p>
            {{ form.as_p   }}
        </p>
        <input type="submit" value="Отправить" >
    </form>
</div>
{% endblock content %}
```

После этих изменений при запуске нашего приложения созданные поля на форме будут представлены в трех видах (рис. 7.56).



Рис. 7.56. Текстовые поля на странице my\_form.html представлены в трех разных видах

В первом случае поля формы представлены в виде таблицы, а все элементы формы выровнены по левому краю. Во втором случае поля формы представлены в виде списка и слева от идентификатора поля присутствует признак элемента списка (маркер). В третьем случае поля формы представлены в виде параграфа (абзаца). При этом идентификаторы полей и сами поля выровнены по левому краю.

### 7.5.6. Проверка (валидация) данных

Теоретически пользователь может ввести в форму и отправить какие угодно данные. Однако не все данные бывают уместными или корректными. Например, в поле для возраста пользователь может ввести отрицательное или четырехзначное число, что вряд ли может считаться корректным возрастом. В Django для проверки корректности вводимых данных предусмотрен *механизм валидации*.

Основными элементами валидации являются правила, которые определяют параметры корректности вводимых данных. Например, для всех полей по умолчанию устанавливается обязательность ввода значения, а при генерации HTML-кода для поля ввода задается атрибут `required` (обязательное значение поля). И когда мы попробуем отправить форму, с отсутствующим значением в каком-либо из ее полей, то получим соответствующее предупреждение. Проверим это на примере. И для начала восстановим код в шаблоне главной страницы `firstapp/my_form.html`, оставив там отображение полей формы в виде таблицы (листинг 7.50).

#### Листинг 7.50. Изменения в файле my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}


<h5>Валидация данных</h5>
<form method="POST">
    {% csrf_token %}
    <table>
        {{ form.as_table }}
    </table>
    <input type="submit" value="Отправить" >
</form>
</div>
{% endblock content %}


```

А в программном коде модуля `forms.py` сформируем четыре поля: Имя, Возраст, Электронный адрес и Согласны получать рекламу (листинг 7.51).

#### Листинг 7.51. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя")
```

```
age = forms.IntegerField(label="Возраст")
email = forms.EmailField(label="Электронный адрес")
reklama = forms.BooleanField(label="Согласны получать рекламу")
```

Если одно из полей в форме окажется незаполненным, то при нажатии на кнопку **Отправить** пользователь получит предупреждение об ошибке (рис. 7.57).

Рис. 7.57. Предупреждение об ошибке (не заполнено текстовое поле)

Таким образом, чтобы отправить данные, пользователь должен будет обязательно ввести какое-либо значение в незаполненное поле. Однако это не всегда нужно, поскольку одни поля всегда должны быть заполнены, а другие — нет. Предположим, что в нашем примере пользователь не желает получать рекламу. Однако он не сможет отправить данные до тех пор, пока не поставит флажок в поле **Согласны получать рекламу** (рис. 7.58).

Рис. 7.58. Невозможность отправки данных при незаполненном поле

Для того чтобы дать пользователю возможность оставить какое-либо поле пустым, необходимо явно отключить для него атрибут `required`. В нашем случае для этого поле `reklama` нужно инициализировать с помощью такого программного кода:

```
reklama = forms.BooleanField(label="Согласны получать рекламу", required=False)
```

Для полей, которые требуют ввода текста, например, CharField, EmailField и подобных, иногда требуется ограничивать число вводимых символов. Это можно сделать с помощью параметров: `max_length` — максимальное число символов и `min_length` — минимальное число символов. Так, в поле, созданное с помощью кода

```
name = forms.CharField(min_length=2, max_length=20)
```

можно будет ввести не менее двух и не более двадцати символов.

По аналогии для числовых полей IntegerField, DecimalField и FloatField можно устанавливать параметры: `max_value` и `min_value`, которые задают соответственно максимально и минимально допустимые значения. Для поля DecimalField дополнительно можно задать еще один параметр — `decimal_places`, который определяет максимальное число знаков после запятой. Так, в поле, созданное с помощью кода

```
age = forms.IntegerField(min_value=1, max_value=120)
weight = forms.DecimalField(min_value=3, max_value=200,
                           decimal_places=2)
```

значение возраста допускается вводить в пределах от 1 до 120 лет, а значение веса — в пределах от 3 до 200 кг. При этом число знаков после десятичной запятой может быть не более двух.

Рассмотренные здесь атрибуты позволяют проверять введенные значения на стороне клиента. Однако возможны варианты, когда пользователи все же смогут отправить форму с заведомо некорректными данными, например воспользовавшись инструментами для разработчиков. Чтобы предупредить такое развитие событий, можно подправить исходный код формы, добавив к ней атрибут `novalidate`, который отключает в веб-браузере проверку данных на стороне клиента, и предусмотреть проверку корректности данных на стороне сервера.

Для этого у формы вызывается метод `is_valid()`, который возвращает `True`, если данные корректны, и `False` — если данные некорректны. Чтобы использовать этот метод, нужно создать объект формы и передать ей данные, пришедшие из запроса.

Рассмотрим пример проверки корректности данных на стороне сервера, для чего изменим программный код модуля `views.py` (листинг 7.52, выделено серым фоном).

#### Листинг 7.52. Изменения в файле `views.py`

```
from django.http import HttpResponse
from django.shortcuts import render
from .forms import UserForm
def index(request):
    my_text = 'Изучаем формы Django'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")

def my_form(request):
    if request.method == "POST":
        userform = UserForm(request.POST)
```

```
if userform.is_valid():
    name = userform.cleaned_data["name"]
    return JsonResponse(
        "<h2>Имя введено корректно – {0} </h2 > ".format(name))
else:
    return JsonResponse("Ошибка ввода данных")
else:
    userform = UserForm()
    return render(request,
                  "firstapp/my_form.html", {"form": userform})
```

Что делает этот программный код? Если выполняется POST-запрос, то вначале происходит заполнение формы пришедшими данными:

```
userform = UserForm(request.POST)
```

Потом с помощью метода `is_valid()` проверяется их корректность:

```
if userform.is_valid():
```

Если данные введены корректно, то через объект `cleaned_data` в переменную `name` заносим введенное пользователем значение и формируем ответную страницу с сообщением, что данные корректны:

```
name = userform.cleaned_data["name"]
return JsonResponse("<h2>Имя введено корректно – {0} </h2 > ".format(name))
```

Если данные введены некорректно, то формируем ответную страницу с сообщением об ошибке:

```
return JsonResponse("Ошибка ввода данных")
```

Если POST-запрос отсутствует, то просто происходит вызов пользовательской формы:

```
userform = UserForm()
return render(request, "firstapp/my_form.html", {"form": userform})
```

Для тестирования нашей формы нужно отключить проверку корректности данных на стороне клиента. Для этого откроем шаблон формы `firstapp\my_form.html` и установим в нем атрибут `novalidate` (листинг 7.53, выделено серым фоном).

#### Листинг 7.53. Изменения в файле my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2 border border-3">
    <h5>Валидация данных</h5>
    <form method="POST" novalidate>
        {% csrf_token %}
        <table>
            {{ form.as_table  }}
        </table>
```

```

<input type="submit" value="Отправить" >
</form>
</div>
{% endblock content %}

```

Внесем в программный код последние корректины — оставим в модуле `forms.py` всего одно поле с именем клиента (листинг 7.54).

#### Листинг 7.54. Изменения в файле forms.py

```

from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя")

```

Итак, все готово. Теперь при запуске приложения мы получим нашу форму с одним полем для ввода данных (рис. 7.59).



Рис. 7.59. Вид формы при незаполненном поле с именем клиента

Оставим поле с именем клиента пустым и нажмем кнопку **Отправить**. Поскольку мы с помощью атрибута `novalidate` искусственно отключили возможность проверки данных на стороне клиента, то пустое поле с именем отправится на сторону сервера, где оно будет обработано с помощью метода `userform.is_valid()`. Если поле оказалось незаполненным, то сервер вернет пользователю страницу с сообщением об ошибке (рис. 7.60).

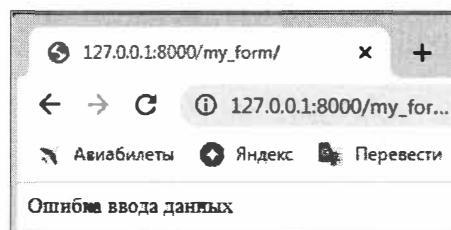


Рис. 7.60. Сообщение об ошибке пользователя со стороны сервера

Теперь заполним поле с именем клиента и нажмем кнопку **Отправить**. Поскольку данные были введены корректно, то метод `userform.is_valid()` вернет пользователю сообщение о корректном вводе данных (рис. 7.61).

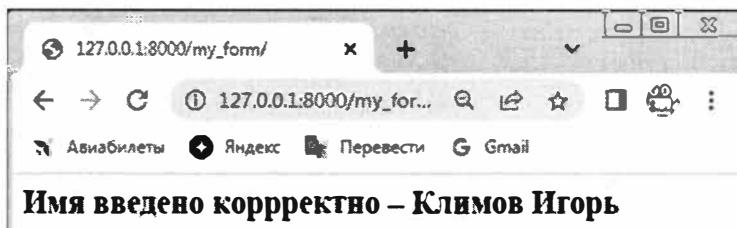


Рис. 7.61. Сообщение о корректности данных, полученное со стороны сервера

### 7.5.7. Детальная настройка полей формы

Формы и поля допускают установку ряда параметров, которые позволяют частично настраивать отображение полей и форм, тем не менее этого бывает недостаточно. Например, необходимо применить стили или добавить рядом с полем ввода какой-нибудь специальный текст. В Django имеется возможность коренным образом изменить всю композицию создаваемых полей.

В частности, в шаблоне компонента мы можем обратиться к каждому отдельному полю формы через название формы: `form.название_поля`. По названию поля мы можем непосредственно получить генерируемый полем HTML-элемент без использования дополнительного кода. Кроме того, каждое поле имеет ряд ассоциированных с ним значений:

- `form.название_поля.name` — возвращает название поля;
- `form.название_поля.value` — возвращает значение поля, которое ему было передано по умолчанию;
- `form.название_поля.label` — возвращает текст метки, которая генерируется рядом с полем;
- `form.название_поля.id_for_label` — возвращает id для поля, которое по умолчанию создается по схеме `id_имя_поля`;
- `form.название_поля.auto_id` — возвращает id для поля, которое по умолчанию создается по схеме `id_имя_поля`;
- `form.название_поля.label_tag` — возвращает элемент `label`, который представляет метку рядом с полем;
- `form.название_поля.help_text` — возвращает текст подсказки, ассоциированной с полем;
- `form.название_поля.errors` — возвращает ошибки валидации, связанные с полем;
- `form.название_поля.css_classes` — возвращает CSS-классы поля;
- `form.название_поля.as_hidden` — генерирует для поля разметку в виде скрытого поля `<input type="hidden">`;
- `form.название_поля.is_hidden` — возвращает True ИЛИ False в зависимости от того, является ли поле скрытым;
- `form.название_поля.as_text` — генерирует для поля разметку в виде текстового поля `<input type="text">`;

- `form.название_поля.as_textarea` — генерирует для поля разметку в виде `<textarea> </textarea>`;
- `form.название_поля.as_widget` — возвращает виджет Django, который ассоциирован с полем.

Так, чтобы, например, получить текст метки поля, которое называется `age`, нам нужно использовать выражение `form.age.label`.

Рассмотрим возможности применения этих значений полей на простом примере, для чего в модуле `forms.py` создадим два поля с именем и возрастом клиента (листинг 7.55).

#### Листинг 7.55. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента")
    age = forms.IntegerField(label="Возраст клиента")
```

В представлении `views.py` передадим эту форму в шаблон с использованием кода листинга 7.56 (изменения выделены серым фоном).

#### Листинг 7.56. Изменения в файле views.py

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import UserForm
def index(request):
    my_text = 'Изучаем формы Django'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
def my_form(request):
    userform = UserForm()
    if request.method == "POST":
        userform = UserForm(request.POST)
        if userform.is_valid():
            name = userform.cleaned_data["name"]
            return HttpResponseRedirect(
                "<h2>Имя введено корректно – {0} </h2 > ".format(name))
    return render(request, "firstapp/my_form.html", {"form": userform})
```

И в шаблоне страницы с нашей формой `firstapp\my_form.html` напишем код использования полей формы (листинг 7.57, изменения выделены серым фоном).

#### Листинг 7.57. Изменения в файле my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
```

```
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2 border border-3">
    <h5>Валидация группы полей</h5>
    <form method="POST" novalidate>
        {% csrf_token %}
        <div class="form-group my-2">
            {% for field in form %}
                <div>{{field.label_tag}}</div>
                <div>{{field}}</div>
                <div class="error">{{field.errors}}</div>
            {% endfor %}
        </div>
        <input type="submit" value="Отправить" >
    </form>
</div>
{% endblock content %}
```

В этом шаблоне форма представляет собой набор полей, расположенных внутри цикла for. С помощью выражения {% for field in form %} в цикле делается проход по всем полям формы, при этом есть возможность управлять отображением, как самих полей, так и связанных с ними атрибутов: ошибок, текста подсказки, меток и т. д.

Здесь мы сгруппировали отображение следующих значений для каждого поля формы:

- field.label\_tag — элемент label, который показывает метку поля;
- field — само поле для ввода данных;
- field.errors — ошибки валидации, связанные с полем.
- Теперь в форме будет отображаться следующая информация: метка поля, под ней само поле. Если в поле пользователь введет ошибочные данные, то информация об ошибке будет отображена под соответствующим полем.

Если мы теперь запустим наше приложение, то получим следующий вид формы (рис. 7.62).

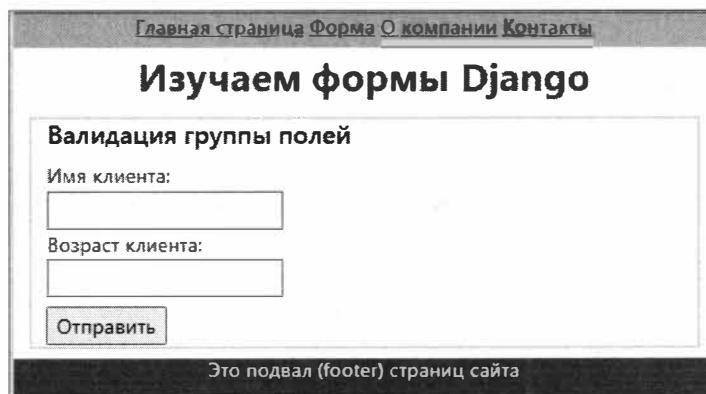


Рис. 7.62. Вид формы при незаполненных полях с данными о клиенте

Если мы теперь при незаполненных полях нажмем кнопку **Отправить**, то возле каждого поля получим сообщение о наличии ошибки (рис. 7.63).

Как видно из рис. 7.63, сообщение о наличии ошибки появилось рядом только с незаполненным полем.

Рис. 7.63. Сообщение об ошибке при незаполненных полях с данными о клиенте

Поле ввода может содержать несколько ошибок. В этом случае можно с помощью тега `for` последовательно вывести все возможные ошибки:

```
{% for error in field.errors %}
    <div class="alert alert-danger">{{error}}</div>
{% endfor %}
```

## 7.5.8. Присвоение стилей полям формы

Поля формы имеют определенные стили по умолчанию. Если же мы хотим применить к ним какие-то другие стили, то для этого нужно использовать ряд заложенных в Django механизмов.

Прежде всего, имеется возможность изменить параметры полей (цвет, размер, форму). В качестве примера возьмем форму с двумя полями, которая задана в модуле `forms.py` (листинг 7.58).

### Листинг 7.58. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента")
    age = forms.IntegerField(label="Возраст клиента")
```

В шаблоне страницы `firstapp\my_form.html`, которая выводит данную форму, добавим код, меняющий стили полей (листинг 7.59, изменения выделены серым фоном).

**Листинг 7.59. Изменения в файле my\_form.html**

```
{% extends "firstapp/base.html" %}  
{% block title %}Формы{% endblock title %}  
{% block header %}Изучаем формы Django{% endblock header %}  
{% block content %}  
<style>  
    .alert{color:red}  
    .form-group{margin:20px;}  
    .form-group input{width:250px; height:25px; border-radius:10px;}  
</style>  
<div class="container-fluid text-start my-2  
      border border-5 border-warning">  
    <h5>Стилизация группы полей</h5>  
    <form method="POST" novalidate>  
        {% csrf_token %}  
        <div class="form-group my-2">  
            {% for field in form %}  
                <div>{{field.label_tag}}</div>  
                <div>{{field}}</div>  
                <div style="color:blue">{{field.help_text}}</div>  
                {% for error in field.errors %}  
                    <div class="alert alert-danger">{{error}}</div>  
                {% endfor %}  
            {% endfor %}  
        </div>  
        <input type="submit" value="Отправить" >  
    </form>  
</div>  
{% endblock content %}
```

Здесь в заголовке head задан стиль alert{color:red} (красный цвет для отображения ошибок) и стили вывода полей, которые объединены в группы:

- margin:20px — отступы;
- width:250px — ширина рамок полей;
- height:25px — высота рамок полей;
- border-radius:10px — радиус округления углов рамок полей.

Далее задан стиль контейнера Bootstrap:

- text-start — прижимать текст в контейнере к левому краю;
- my-2 — вертикальные отступы между элементами контейнера 2 пункта;
- border border-5 — ширина рамки контейнера 5 пунктов;
- border-warning — цвет рамки контейнера (warning).

В блоке с полями формы задан индивидуальный стиль для вывода подсказок шрифтом синего цвета (style="color:blue"). Наконец, ошибки выдаются в теге с циклом {% for error in field.errors %} с использованием специального класса Bootstrap — class="alert alert-danger".

Если после этих изменений запустить наше приложение, то вся форма будет выглядеть так, как показано на рис. 7.64.

На данном рисунке видно, что рамки имеют округлую форму, а подсказки выдаются синим цветом.

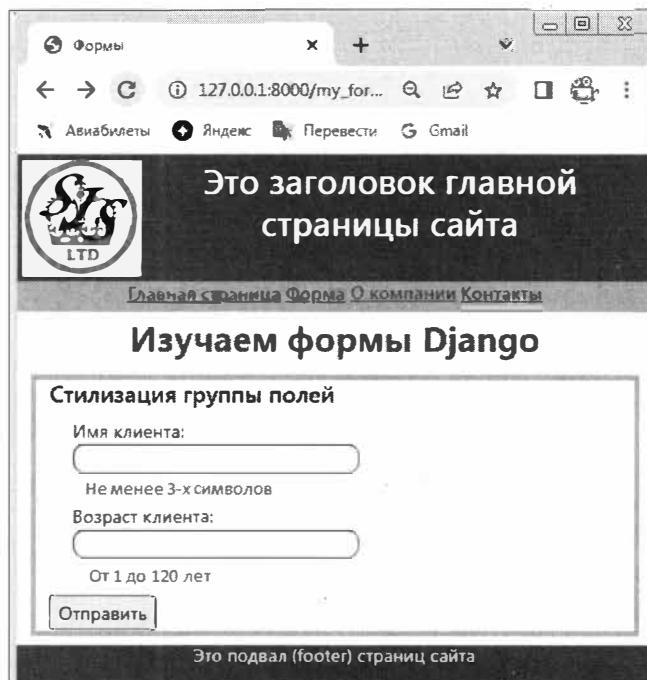


Рис. 7.64. Шаблон формы my\_form.html после стилизации полей

Если теперь попытаться отправить на сервер содержимое формы с пустыми полями, то будут выведены сообщения об ошибках с текстом красного цвета (рис. 7.65).

В Django имеется еще один механизм для стилизации форм — через свойства формы: `required_css_class` и `error_css_class`. Эти свойства применяют CSS-класс к метке, создаваемой для поля формы, и к строке вывода сообщений об ошибках. Рассмотрим применение этого механизма на следующем примере. Для этого изменим модуль `forms.py`, как показано в листинге 7.60 (выделено серым фоном).

#### Листинг 7.60. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента", min_length=3,
                           help_text='Не менее 3 символов')
    age = forms.IntegerField(label="Возраст клиента",
                            min_value=1, max_value=120,
                            help_text='От 1 до 120 лет')
    required_css_class = "field"
    error_css_class = "error"
```

The screenshot shows a Django form titled "Изучаем формы Django". The form has two input fields: "Имя клиента:" containing "Климов Игорь" and "Возраст клиента:". A tooltip-like callout bubble labeled "Сообщение об ошибке" points to the "Возраст клиента:" field, which contains "От 1 до 120 лет". Below the fields is a message box with a gray background and white text: "Обязательное поле.". At the bottom is a "Отправить" button and a footer bar with the text "Это подвал (footer) страниц сайта".

Рис. 7.65. Сообщение об ошибках в форме со стилизованными полями

После этого в шаблоне формы `firstapp\my_form.html` нужно подключить классы `field` и `error` и задать им желаемые свойства, как показано в листинге 7.61 (изменения выделены серым фоном).

#### Листинг 7.61. Изменения в файле `my_form.html`

```
{% extends "firstapp/base.html" %}  
{% block title %}Формы{% endblock title %}  
{% block header %}Изучаем формы Django{% endblock header %}  
{% block content %}  
<style>  
    .field{font-weight:bold; color:darkblue}  
    .error{color:red}  
</style>  
<div class="container-fluid text-start my-2  
      border border-5 border-warning">  
    <h5>Стилизация группы полей</h5>  
    <form method="POST" novalidate>  
        {% csrf_token %}  
        <div class="form-group my-2">  
            {% for field in form %}  
                <div>{{field.label_tag}}</div>  
                <div>{{field}}</div>  
                <div class="error">{{field.errors}}</div>  
            {% endfor %}  
        </div>  
        <input type="submit" value="Отправить" >  
    </form>  
</div>  
{% endblock content %}
```

В этом программном коде для меток полей задан полужирный шрифт (`font-weight:bold`) темно-синего цвета (`color:darkblue`). Для сообщений об ошибках задан красный цвет шрифта — `error{color:red}`.

Если после этих изменений запустить наше приложение, то будет видно, что метки полей выводятся полужирным шрифтом темно-синего цвета, а сообщения об ошибках выводятся красным цветом (рис. 7.66).

The screenshot shows a Django form titled "Изучаем формы Django". It contains two fields: "Имя клиента:" with the value "Климов Игорь" and "Возраст клиента:". Below the first field is a callout bubble labeled "Метка поля" (Field Label). Below the second field is another callout bubble labeled "Сообщение об ошибке" (Error Message) with the text "• Обязательное поле." (Required field). At the bottom is a "Отправить" (Send) button. A footer at the bottom of the page reads "Это подвал (footer) страниц сайта" (This is the footer of the website page).

Рис. 7.66. Сообщение об ошибках в форме с присвоенными стилями

В Django имеется и третий механизм присвоения стилей формам — через установку стилей в виджетах. Рассмотрим пример. Изменим кодовые строки модуля `forms.py`, создающие поля (листинг 7.62).

#### Листинг 7.62. Изменения в файле forms.py

```
from django import forms
class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента",
                           widget=forms.TextInput(attrs={"class": "myfield"}))
    age = forms.IntegerField(label="Возраст клиента",
                            widget=forms.NumberInput(attrs={"class": "myfield"}))
```

В этом коде через параметр виджетов `attrs` задаются атрибуты того HTML-элемента, который будет генерироваться. В частности, здесь для обоих полей устанавливается атрибут `class`, который представляет класс `myfield`.

Теперь нужно в шаблоне главной страницы `firstapp\my_form.html` определить класс `myfield` и задать для него набор свойств, как показано в листинге 7.63 (выделено серым фоном).

#### Листинг 7.63. Изменения в файле my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
```

```
<style>
    .myfield{border:2px solid #ccc;
              border-radius:5px;
              height:25px;
              width:200px;
              margin: 10px 10px 10px 0}
</style>
<div class="container-fluid text-start my-2
          border border-5 border-warning">
    <h5>Стилизация группы полей</h5>
    <form method="POST" novalidate>
        {% csrf_token %}
        <div class="form-group my-2">
            {% for field in form %}
                <div>{{field.label_tag}}</div>
                <div>{{field}}</div>
                <div class="error">{{field.errors}}</div>
            {% endfor %}
        </div>
        <input type="submit" value="Отправить" >
    </form>
</div>
{% endblock content %}
```

Если после этих изменений запустить наше приложение, то форма будет выведена в следующем виде (рис. 7.67).

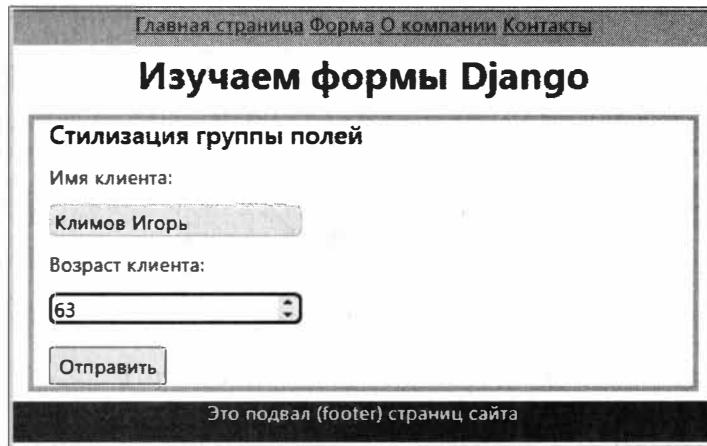


Рис. 7.67. Вид формы со стилями, присвоенными через параметры виджетов

## 7.6. Использование в формах POST-запросов для отправки данных на сервер

В форму, которую мы создали в предыдущем разделе, пользователь может не только ввести свои данные, но затем отправить их на сервер. Уберем параметры стилизации,

оставим проверку ошибок пользователя на стороне браузера и более детально рассмотрим шаблон формы `my_form.html`, которая теперь будет иметь вид, представленный в листинге 7.64.

#### Листинг 7.64. Изменения в файле `my_form.html`

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2
border border-5 border-warning">
<h5>Стилизация группы полей</h5>
<form method="POST">
    {% csrf_token %}
    <div class="form-group my-2">
        {% for field in form %}
            <div>{{field.label_tag}}</div>
            <div>{{field}}</div>
            <div class="error">{{field.errors}}</div>
        {% endfor %}
    </div>
    <input type="submit" value="Отправить" >
</form>
</div>
{% endblock content %}
```

Для создания формы здесь использован стандартный элемент HTML `<form>`. В начале формы помещен встроенный тег Django `{% csrf_token %}`, который позволяет защитить приложение от CSRF-атак, добавляя в форму csrf-токен в виде скрытого поля. В нижней части формы помещена кнопка для отправки содержимого этой формы на сервер.

#### **ПОЯСНЕНИЕ**

CSRF (Cross-Site Request Forgery, также XSRF) — опаснейшая атака, которая приводит к тому, что хакер может выполнить на неподготовленном сайте массу различных действий от имени других зарегистрированных посетителей.

Далее в представлении (в файле `hello/firstapp/views.py`) немного изменим и разберем код функции `my_form()` (листинг 7.65, здесь функция выделена серым фоном).

#### Листинг 7.65. Изменения в файле `views.py`

```
# python manage.py runserver
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import UserForm
def index(request):
    my_text = 'Изучаем формы Django'
    context = {'my_text': my_text}
    return render(request, "firstapp/index.html", context)
```

```
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
def my_form(request):
    if request.method == "POST":
        userform = UserForm(request.POST)
        if userform.is_valid():
            name = request.POST.get("name") # получить значение поля Имя
            age = request.POST.get("age") # получить значение поля Возраст
            output = "<h2>Пользователь</h2><h3>Имя - {0}, "
            " Возраст - {1} </h3 >".format(name, age)
        return HttpResponseRedirect(output)
    userform = UserForm()
    return render(request, "firstapp/my_form.html", {"form": userform})
```

Разберемся, какие действия запрограммированы в этом коде. Для отправки данных на сервер обычно используется запрос типа POST. Для определения типа запроса делается проверка значения запроса `request.method` в структуре `if...else`.

Если запрос не соответствует типу POST (ветка else), то мы просто формируем пользовательскую форму `userform` и отправляем ее для ввода данных в шаблон `my_form.html`. Таким образом, при первом обращении к приложению мы вначале увидим сформированную нами форму ввода. Введем в нее некоторые данные (рис. 7.68).

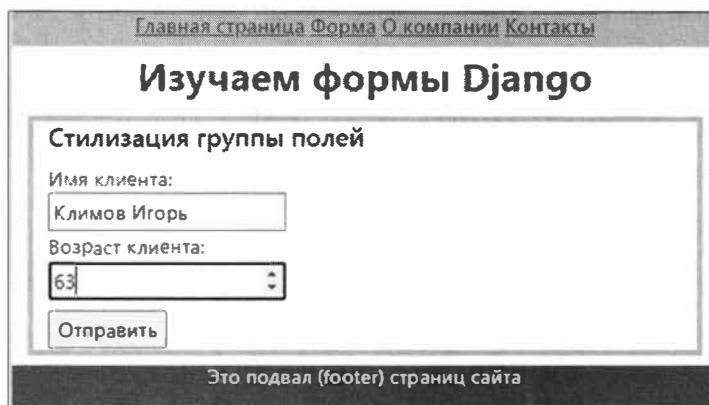


Рис. 7.68. Ввод данных в форму при первой загрузке страницы index.html

Если теперь нажать на кнопку **Отправить**, то запрос будет иметь тип POST (`request.method == "POST"`), это значит, что данные отправляются на сервер. В этом случае отправляемые из формы данные присваиваются переменным `name` и `age`, а их значения — переменной `output`. После этого значения из `output` отправляются на сервер через объект `HttpResponse`. На сервере эти данные могут быть либо записаны в БД, либо обработаны одной из процедур, которая выполнит определенные действия. Поскольку мы еще не изучали возможности работы Django с БД, в этом примере ответ отправляется пользователю на ту же HTML-страницу (рис. 7.69).

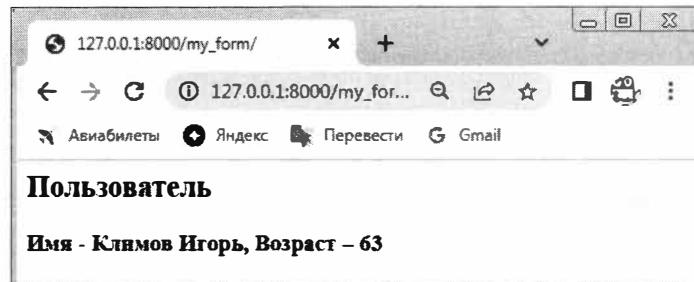


Рис. 7.69. Ответ из формы по нажатии кнопки Отправить

Конечно, мы здесь реализовали простейший вариант, когда все действия совершаются в пределах одной страницы. В реально работающих приложениях выполняется либо переадресация данных (отправка на другую страницу), либо их запись в базу данных, что мы и сделаем в последующих разделах.

## 7.7. Краткие итоги

В этой главе были приведены сведения о формах. Мы узнали, как создать форму, как определить, настроить и стилизовать в форме поля, как проверить корректность введенных в форму данных. Теперь нам нужно разобраться с тем, как Django работает с самими данными. Понять, как можно сохранить в БД данные, которые пользователь ввел в форме, или как получить из БД данные, запрошенные пользователем через свой веб-браузер. В Django все действия с информацией, которая хранится в БД, осуществляются через так называемые *модели данных*, или, просто, *модели*. Изучению моделей данных посвящена следующая глава. В ней рассматриваются методы работы с данными через модели, не прибегая к SQL-запросам — традиционному способу взаимодействия с СУБД.



## ГЛАВА 8

# Модели данных Django

Практически любое веб-приложение, так или иначе, работает с базой данных. При этом с системой управления базами данных (СУБД) приложение общается каким-нибудь универсальным способом, например посредством языка SQL. Однако программисту чаще всего хочется иметь некую абстракцию, позволяющую большую часть времени работать с привычными сущностями языка. Такой абстракцией является Object-Relational Mapping (ORM) — отображение сущностей предметной области и их взаимосвязей в объектах, удобных для использования программистом.

Имеются разные подходы к тому, как нужно изолировать пользователя от конкретного хранилища данных. Есть такие, которые полностью скрывают всю работу с БД, — вы пользуетесь объектами, изменяете их состояние, а ORM неявно синхронизирует состояние объектов и сущностей в хранилище. Другие ORM всего лишь обрабатывают сущности БД в структуры языка, но все запросы нужно писать вручную. Это два разных полюса, каждый со своими плюсами и минусами. Авторы Django решили остаться где-то посередине.

В Django вы работаете с объектами и выполняете вручную их загрузку и сохранение, однако используете для этого привычные средства языка — вызовы методов и свойств. При этом Django берет на себя обеспечение правильной работы вашего приложения с конкретными хранилищами данных. Эта изоляция от конкретного хранилища позволяет применять разные БД в различных условиях: при разработке и тестировании задействовать легковесные СУБД, а при развертывании сайтов в реальных условиях переходить на более мощные и производительные базы данных.

Термин «модель» часто употребляется в качестве замены словосочетания «Django ORM», поэтому мы будем им пользоваться и в рамках этой книги. «Модель» говорит нам о том, что наша предметная область смоделирована с помощью средств фреймворка. При этом отдельные сущности тоже называются *моделями* — модели поменьше собираются в большую «Модель» всей предметной области.

Связь между моделями и таблицами в БД максимально прямая: одна таблица — одна модель. В этом плане Django ORM не отходит далеко от схемы БД — вы всегда имеете представление о том, как фактически описаны ваши данные с точки зрения СУБД. Что очень полезно, когда нужно что-либо где-то оптимизировать!

Из материалов этой главы вы узнаете:

- какие типы полей можно использовать в модели данных Django;
- какие существуют методы для работы с данными в Django (добавление, чтение, обновление, удаление данных из БД);
- как можно манипулировать с данными на конкретных примерах работы с объектами модели данных;
- как организовать различные типы связей между таблицами в модели данных.

## 8.1. Создание моделей и миграции базы данных

Модели в Django описывают структуру используемых в программе данных. Эти данные хранятся в базах данных, и с помощью моделей как раз осуществляется взаимодействие с такими базами.

По умолчанию Django в качестве базы данных задает SQLite. Она очень проста в использовании и не требует запущенного сервера. Все файлы базы данных можно легко переносить с одного компьютера на другой. Однако при необходимости мы можем использовать в Django множество других распространенных СУБД.

Для работы с базами данных в проекте Django в файле `settings.py` определен параметр `DATABASES`, который по умолчанию выглядит следующим образом:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

Конфигурация базы данных в таком случае складывается из двух параметров. Параметр `ENGINE` указывает на применяемый для доступа к БД движок. В нашем случае это встроенный пакет `django.db.backends.sqlite3`. Второй параметр — `NAME` указывает на имя и путь к базе данных. После первого запуска проекта в нем по умолчанию будет создан файл `db.sqlite3`, который, собственно, и будет служить в качестве базы данных.

Для работы с другими системами управления базами данных необходимо установить соответствующий пакет (табл. 8.1).

**Таблица 8.1. Дополнительные пакеты для работы Django с различными СУБД**

СУБД	Пакет	Команда установки
PostgreSQL	<code>psycopg2</code>	<code>pip install psycopg2</code>
MySQL	<code>mysql-python</code>	<code>pip install mysql-python</code>
Oracle	<code>cx_Oracle</code>	<code>pip install cx_Oracle</code>

При создании приложения по умолчанию в его каталог добавляется файл `models.py`, который применяется для определения и описания моделей. Модель представляет собой класс, унаследованный от `django.db.models.Model`.

Рассмотрим процесс создания модели на простом примере и создадим модель, описывающую клиента, имеющего две характеристики: имя (name) и возраст (age). Изменим файл models.py, как показано в листинге 8.1 и на рис. 8.1.

#### Листинг 8.1. Файл models.py

```
from django.db import models
class Person(models.Model):
    name = models.CharField(max_length=20)
    age = models.IntegerField()
```

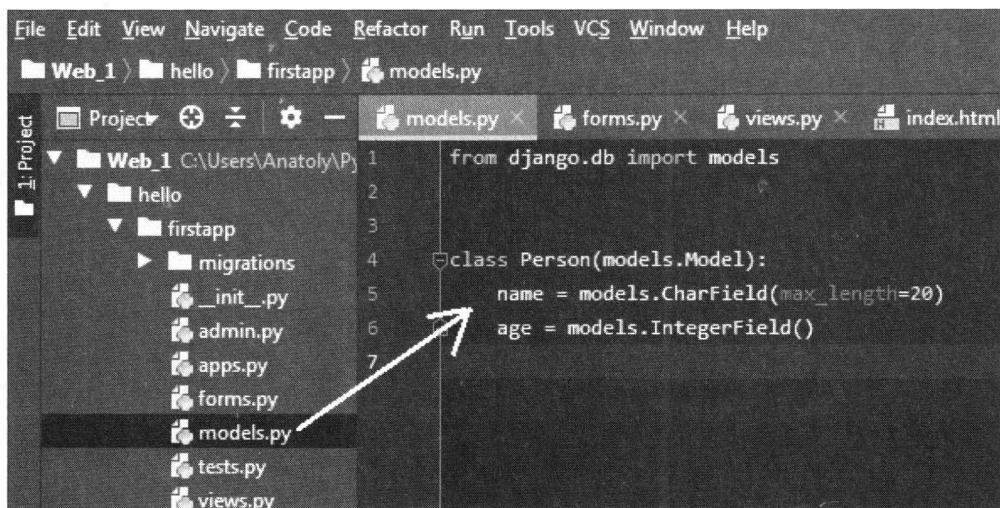


Рис. 8.1. Добавление кода в файл models.py приложения firstapp

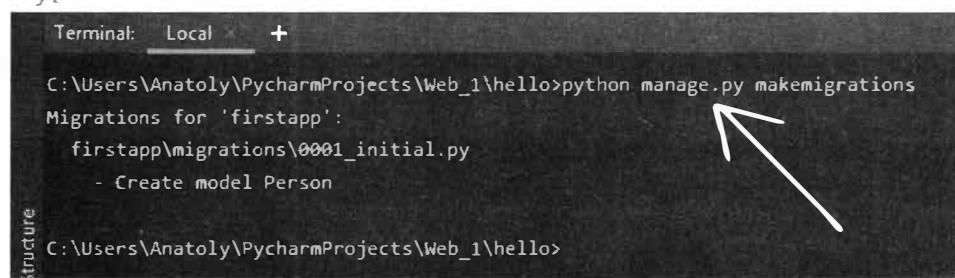
Здесь в программном коде определена простейшая модель с именем Person, представляющая характеристики человека (клиента системы). В модели определены два поля для хранения имени и возраста. Поле name представляет тип CharField — текстовое поле, которое хранит последовательность символов. В нашем случае — имя клиента. Для CharField указан параметр max\_length, задающий максимальную длину хранящейся строки. Поле age представляет тип IntegerField — числовое поле, которое содержит целые числа. Оно предназначено для хранения возраста человека.

Каждая модель сопоставляется с определенной таблицей в базе данных. Однако пока у нас нет в БД ни одной таблицы, которая хранила бы объекты модели Person. На следующем шаге нам нужно создать такую таблицу. В Django это делается с помощью *миграции* — процесса внесения изменений в базу данных в соответствии с определениями в модели.

Миграция формируется в два шага. На первом шаге необходимо создать миграцию (файл с параметрами миграции) с помощью команды

```
python manage.py makemigrations
```

Откроем окно терминала PyCharm и выполним эту команду (рис. 8.2).



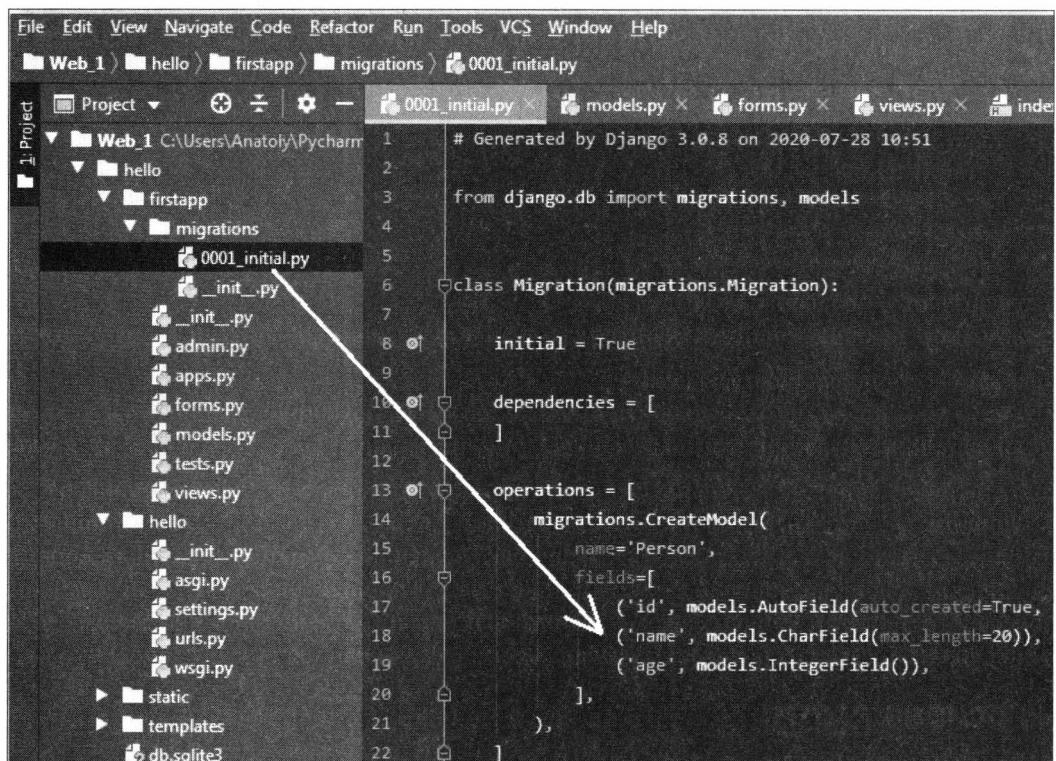
```
Terminal: Local +
```

```
C:\Users\Anatoly\PycharmProjects\Web_1\hello>python manage.py makemigrations
Migrations for 'firstapp':
  firstapp\migrations\0001_initial.py
    - Create model Person
```

Structure  
C:\Users\Anatoly\PycharmProjects\Web\_1\hello>

Рис. 8.2. Создание миграции модели данных в окне терминала PyCharm

Как можно видеть, после выполнения этой команды создан файл с миграцией `firstapp\migrations\0001_initial.py` и получено сообщение: **Create model Person** (создана модель Person).



File Edit View Navigate Code Refactor Run Tools VCS Window Help

Web\_1 > hello > firstapp > migrations > 0001\_initial.py

Project

```
1 # Generated by Django 3.0.8 on 2020-07-28 10:51
2
3 from django.db import migrations, models
4
5
6 class Migration(migrations.Migration):
7     initial = True
8
9     dependencies = [
10         ]
11
12     operations = [
13         migrations.CreateModel(
14             name='Person',
15             fields=[
16                 ('id', models.AutoField(auto_created=True)),
17                 ('name', models.CharField(max_length=20)),
18                 ('age', models.IntegerField()),
19             ],
20         ),
21     ]
22 ]
```

Рис. 8.3. Окно PyCharm: описание полей модели данных в файле миграции `0001_initial.py`

Новый файл `0001_initial.py`, расположенный в папке `migrations` нашего приложения `firstapp`, будет иметь примерно такое содержимое (рис. 8.3):

```
from django.db import migrations, models
class Migration(migrations.Migration):
    initial = True
    dependencies = [
    ]
```

```
operations = [
    migrations.CreateModel(
        name='Person',
        fields=[

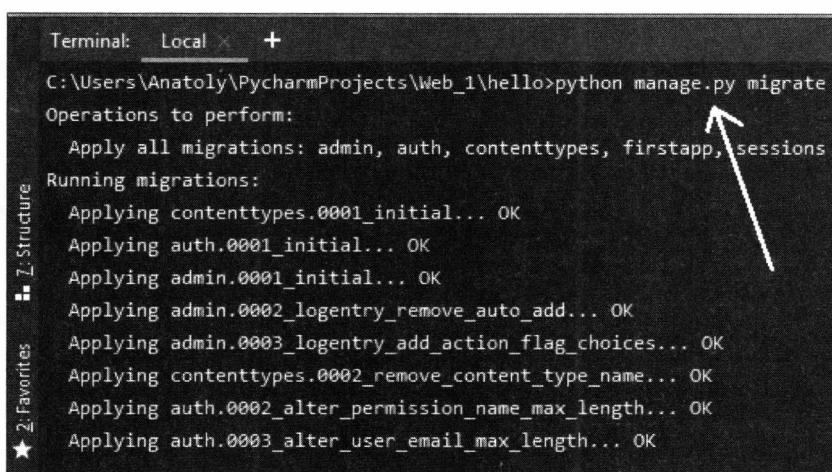
            ('id', models.AutoField(auto_created=True,
                                   primary_key=True, serialize=False,
                                   verbose_name='ID')),
            ('name', models.CharField(max_length=20)),
            ('age', models.IntegerField()),
        ],
    ),
]
```

Это и есть результаты первого шага миграции. Здесь можно заметить, что в процессе миграции создано не два поля, как было описано в модели, а три — появилось дополнительное поле `id`, которое будет представлять первичный ключ в таблице базы данных. Такое поле добавляется автоматически по умолчанию. Поэтому в Django в самой модели не требуется для таблиц базы данных явным образом определять ключевые поля — они будут создаваться автоматически. На первом шаге было создано только описание полей для таблицы `Person`, в которой будут храниться данные о клиентах, при этом сама таблица в БД еще отсутствует.

На втором шаге миграции необходимо на основе описания полей в модели данных создать соответствующую таблицу в БД. Это делается с помощью команды

```
python manage.py migrate
```

Откроем окно терминала PyCharm и выполним эту команду (рис. 8.4).

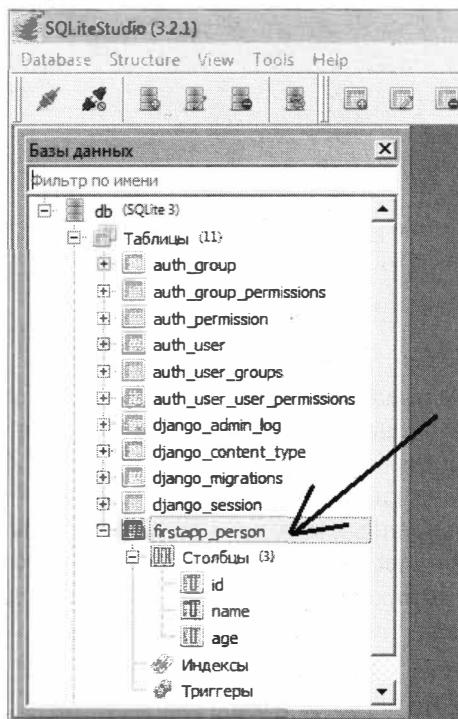


The screenshot shows a PyCharm terminal window titled "Local". The command "python manage.py migrate" is entered, followed by the output of the migration process:

```
C:\Users\Anatoly\PycharmProjects\Web_1\hello>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, firstapp, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
```

Рис. 8.4. Окно терминала PyCharm: создание таблиц в БД при миграции модели данных

Теперь, если мы откроем присутствующую в проекте базу данных `db.sqlite3` с помощью программы для просмотра БД (например, SQLiteStudio), то увидим, что она содержит ряд таблиц (рис. 8.5). В основном это служебные таблицы. Но нас интересует наша таблица `Person`, в которой будут храниться данные о клиентах. В Django имена таблиц



**Рис. 8.5.** Окно приложения SQLiteStudio:  
таблица firstapp\_person,  
созданная в БД при миграции  
модели данных

формируются автоматически, при этом имя таблицы состоит из двух частей: из имени приложения и имени модели. В нашем случае таблица будет иметь имя `firstapp_person`.

## 8.2. Типы полей в модели данных Django

В Django для определения моделей данных предусмотрены следующие типы полей:

- `AutoField()` — хранит целочисленное значение, которое автоматически инкрементируется (увеличивается на 1). Обычно применяется для первичных ключей;
- `BigAutoField` — 64-разрядное целое число, похожее на `AutoField`, за исключением того, что оно гарантированно соответствует числам от 1 до 9 223 372 036 854 775 807;
- `BigIntegerField()` — представляет целое число (значение типа `Number`), которое укладывается в диапазон от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 (в зависимости от выбранной СУБД диапазон может немного отличаться);
- `BinaryField()` — хранит бинарные (двоичные) данные;
- `BooleanField()` — хранит значение `True` или `False` (0 или 1);
- `CharField(max_length=N)` — хранит строку длиной не более N символов;
- `DateField()` — хранит дату;
- `DateTimeField()` — хранит дату и время;
- `DecimalField(decimal_places=X, max_digits=Y)` — представляет значение числа с дробной частью (типа `Number`), которое имеет максимум X разрядов и Y знаков после запятой;

- DurationField() — хранит период времени (интервал времени);
- EmailField() — хранит строку, которая представляет email-адрес (значение автоматически проверяется встроенным валидатором EmailValidator);
- FileField() — хранит строку, которая представляет имя файла;
- FilePathField() — хранит строку, которая представляет путь к файлу длиной в 100 символов;
- FloatField() — хранит значение типа Number, которое представляет число с плавающей точкой;
- GenericIPAddressField() — хранит строку, которая представляет IP-адрес в формате IPv4 или IPv6;
- ImageField() — хранит строку, которая представляет данные об изображении;
- IntegerField() — хранит значение типа Number, которое представляет целочисленное значение;
- JSONField() — поле для хранения данных в кодировке JSON;
- PositiveBigIntegerField() — хранит значение типа Number, которое представляет положительное целочисленное значение (от 0 до 9 223 372 036 854 775 807);
- PositiveIntegerField() — хранит значение типа Number, которое представляет положительное целочисленное значение (от 0 до 2 147 483 647);
- PositiveSmallIntegerField() — хранит значение типа Number, которое представляет небольшое положительное целочисленное значение (от 0 до 32 767);
- SlugField() — хранит строку, которая может содержать только буквы в нижнем регистре, цифры, дефис и знак подчеркивания;
- SmallAutoField — подобно AutoField, но допускает значения в определенном, зависящем от базы данных диапазоне (от 1 до 32 767);
- SmallIntegerField() — хранит значение типа Number, которое представляет небольшое целочисленное значение (от -32 768 до 32 767);
- TextField() — хранит строку неопределенной длины;
- TimeField() — хранит время;
- URLField() — хранит строку, которая представляет валидный URL-адрес;
- UUIDField() — хранит строку, которая представляет UUID-идентификатор.

Кроме полей моделей, которые используются для задания типа поля для хранения данных в таблице БД, в Django есть набор полей, которые задают отношения:

- ForeignKey() — отношения «многие к одному»;
- ManyToManyField() — отношения «многие ко многим»;
- OneToOneField() — отношения «один к одному».

Сопоставление типов полей в моделях данных Django с типами полей в различных СУБД приведено в табл. 8.2.

**Таблица 8.2. Сопоставление типов полей в моделях данных Django с типами полей в различных СУБД**

Тип	SQLite	MySQL	PostgreSQL	Oracle
AutoField()	integer NOT NULL AUTOINCREMENT	integer AUTO_INCREMENT NOT NULL	serial NOT NULL	NUMBER(11) NOT NULL
BigIntegerField	bigint NOT NULL	bigint NOT NULL	bigint NOT NULL	NUMBER(19) NOT NULL
BinaryField()	BLOB NOT NULL	longblob NOT NULL	bytea NOT NULL	BLOB NULL
BooleanField()	bool NOT NULL	bool NOT NULL	boolean NOT NULL	NUMBER(1) NOT NULL CHECK ("Значение" IN(0,1))
CharField(max_length=N)	varchar(N) NOT NULL	varchar(N) NOT NULL	varchar(N) NOT NULL	NVARCHAR2(N) NULL
DateField()	date NULL	date NULL	date NULL	DATE NOT NULL
DateTimeField()	datetime NULL	datetime NULL	timestamp NULL	TIMESTAMP NOT NULL
DecimalField(decimal_places=X, max_digits=Y)	decimal NOT NULL	numeric(X, Y) NOT NULL	numeric(X, Y) NOT NULL	NUMBER(10, 3) NOT NULL
DurationField()	bigint NOT NULL	bigint NOT NULL	interval NOT NULL	INTERVAL DAY(9) TO SECOND(6) NOT NULL
EmailField()	varchar(254) NOT NULL	varchar(254) NOT NULL	varchar(254) NOT NULL	NVARCHAR2(254) NULL
FileField()	varchar(100) NOT NULL	varchar(100) NOT NULL	varchar(100) NOT NULL	NVARCHAR2(100) NULL
FilePathField()	varchar(100) NOT NULL	varchar(100) NOT NULL	varchar(100) NOT NULL	NVARCHAR2(100) NULL
FloatField	real NOT NULL	double precision NOT NULL	double precision NOT NULL	DOUBLE PRECISION NOT NULL
GenericIPAddressField()	char(39) NOT NULL	char(39) NOT NULL	inet NOT NULL	VARCHAR2(39) NULL
ImageField()	varchar(100) NOT NULL	varchar(100) NOT NULL	varchar(100) NOT NULL	NVARCHAR2(100) NULL
IntegerField	integer NOT NULL	integer NOT NULL	integer NOT NULL	NUMBER(11) NOT NULL
PositiveIntegerField	integer unsigned NOT NULL	integer UNSIGNED NOT NULL	integer NOT NULL CHECK ("Значение" > 0)	NUMBER NOT NULL CHECK ("Значение" > 0)
PositiveSmallIntegerField	smallint unsigned NOT NULL	smallint UNSIGNED NOT NULL	smallint NOT NULL CHECK ("Значение" > 0)	NUMBER(11) NOT NULL CHECK ("Значение" > 0)
SlugField()	varchar(50) NOT NULL	varchar(50) NOT NULL	varchar(50) NOT NULL	NVARCHAR2(50) NULL
SmallIntegerField	smallint NOT NULL	smallint NOT NULL	smallint NOT NULL	NUMBER(11) NOT NULL

Таблица 8.2 (окончание)

Тип	SQLite	MySQL	PostgreSQL	Oracle
TextField()	text NOT NULL	longtext NOT NULL	text NOT NULL	NCLOB NULL
TimeField()	time NULL	time NULL	time NULL	TIMESTAMP NOT NULL
URLField()	varchar(200) NOT NULL	varchar(200) NOT NULL	varchar(200) NOT NULL	NVARCHAR2(200) NULL
UUIDField()	char(32) NOT NULL	char(32) NOT NULL	uuid NOT NULL	VARCHAR2(32) NULL

## 8.3. Манипуляция с данными в Django на основе CRUD

Аббревиатура CRUD обозначает четыре основные операции, которые используются при работе с базами данных. Этот термин представляет собой сочетание первых букв английских слов: создание (Create), чтение (Read), модификация (Update) и удаление (Delete). Это по своей сути стандартная классификация функций для манипуляций с данными. В SQL этим функциям соответствуют операторы `Insert` (создание записей), `Select` (чтение записей), `Update` (редактирование записей) и `Delete` (удаление записей).

В Django при создании моделей данных они наследуют свое поведение от класса `django.db.models.Model`, который предоставляет базовые операции с данными (добавление, чтение, обновление и удаление).

Рассмотрим выполнение этих операций с данными на примере модели `Person`, которая была создана в предыдущих разделах:

```
from django.db import models
class Person(models.Model):
    name = models.CharField(max_length=20)
    age = models.IntegerField()
```

### 8.3.1. Добавление данных в БД

В Django для добавления данных в БД предусмотрены два метода: `create()` (создать) и `save()` (сохранить).

Добавить данные методом `create()` для нашей модели можно с помощью следующего кода:

```
igor = Person.objects.create(name="Игорь", age=23)
```

Если добавление пройдет успешно, то объект будет иметь `id`, который можно получить через `igor.id`.

Однако, по существу, метод `create()` вызывает другой метод — `save()`, и его мы также можем использовать самостоятельно для добавления объекта в БД:

```
igor = Person(name="Игорь", age=23)
igor.save()
```

После успешного добавления данных в БД можно аналогично получить идентификатор добавленной записи — через `igor.id` и сами добавленные значения — через `igor.name` и `igor.age`.

### 8.3.2. Чтение данных из БД

В Django получить значение данных из БД можно несколькими методами:

- `get()` — для одного объекта;
- `get_or_create()` — для одного объекта с добавлением его в БД;
- `all()` — для всех объектов;
- `count()` — получить число объектов;
- `filter()` — для группы объектов по фильтру;
- `exclude()` — для группы объектов с исключением некоторых;
- `in_bulk()` — для группы объектов в виде словаря.

Методы `all()`, `filter()` и `exclude()` возвращают объект `QuerySet`. Это, по сути, некое промежуточное хранилище, в котором содержится информация, полученная из БД. Объект `QuerySet` может быть создан, отфильтрован и затем использован фактически без выполнения запросов к базе данных. База данных не будет затронута, пока вы не инициируете новое выполнение `QuerySet`. Рассмотрим последовательно все указанные методы.

#### Метод `get()`

Метод `get()` возвращает один объект, т. е. одну запись из БД по определенному условию, которое передается в качестве параметра. Пусть, например, в программе имеется следующий код:

```
klient1 = Person.objects.get(name="Виктор")
klient2 = Person.objects.get(age=25)
klient3 = Person.objects.get(name="Василий", age=23)
```

В этом случае в переменную `klient1` будет считан объект, у которого поле `name="Виктор"`, в переменную `klient2` — объект, у которого поле `age=25`. Соответственно в переменную `klient3` будет считан объект, у которого поле `name="Василий"` и поле `age=23`.

При использовании этого метода нужно учитывать, что он предназначен для выборки таких объектов, которые имеются в базе данных в единственном числе. Если в таблице не окажется подобного объекта, то будет выдана ошибка `имя_модели.DoesNotExist`. Если же в таблице присутствует несколько объектов, которые соответствуют указанному условию, то будет сгенерировано исключение `MultipleObjectsReturned`. Поэтому следует применять этот метод с достаточной осторожностью.

#### Метод `get_or_create()`

Метод `get_or_create()` получает объект из БД, а если его там нет, то он будет добавлен в БД как новый объект. Рассмотрим следующий код:

```
bob, created = Person.objects.get_or_create(name="Bob", age=24)
print(bob.name)
print(bob.age)
```

Этот код вернет добавленный в БД объект (в нашем случае — переменную `bob`) и булево значение (`created`), которое будет иметь значение `True`, если добавление прошло успешно.

## Метод `all()`

Метод `all()` позволяет получить все имеющиеся объекты из базы данных, например:

```
people = Person.objects.all()
```

## Метод `count()`

Метод `count()` позволяет получить число объектов в таблице базы данных, например:

```
people = Person.objects.count()
```

## Метод `filter()`

Если требуется получить объекты, которые соответствуют определенному критерию, то применяется метод `filter()`, который в качестве параметра принимает критерий выборки. Например:

```
people = Person.objects.filter(age=23)
people2 = Person.objects.filter(name="Tom", age=23)
```

Здесь в `people` будут помещены все объекты из БД, для которых `age=23`, а в `people2` — все объекты с параметрами `name="Tom"` и `age=23`.

## Метод `exclude()`

Метод `exclude()` позволяет выбрать из БД все записи, за исключением тех, которые соответствуют критерию, переданному в качестве параметра:

```
people = Person.objects.exclude(age=23)
```

Здесь в `people` будут помещены все объекты из БД, за исключением тех, у которых `age=23`.

Можно комбинировать методы `exclude()` и `filter()`:

```
people = Person.objects.filter(name="Tom").exclude(age=23)
```

Здесь в `people` будут помещены все объекты из БД с именем `name="Tom"`, за исключением тех, у которых `age=23`.

## Метод `in_bulk()`

Метод `in_bulk()` является наиболее эффективным способом для чтения большого количества записей. Он возвращает словарь, т. е. объект `dict`, тогда как методы `all()`, `filter()` и `exclude()` возвращают объект `QuerySet`.

Все объекты из БД можно получить с помощью следующего кода:

```
people = Person.objects.in_bulk()
```

Для получения доступа к одной из выбранных из БД записей нужно указать идентификатор записи в словаре:

```
for id in people:
    print(people[id].name)
    print(people[id].age)
```

С помощью метода `in_bulk()` можно получить и часть объектов из БД. Например, в следующем программном коде из БД будут получены только те объекты, у которых ключевые значения полей равны 1 и 3:

```
people2 = Person.objects.in_bulk([1,3])
for id in people2:
    print(people2[id].name)
    print(people2[id].age)
```

Здесь метод `in_bulk` возвращает словарь, где ключи представляют `id` объектов, а значения по этим ключам — собственно эти объекты, т. е. в нашем случае объекты `Person`.

### 8.3.3. Обновление данных в БД

Для обновления объекта в БД применяется метод `save()`. При этом Django полностью обновляет объект и все его свойства, даже если мы их не изменяли. Например:

```
nic = Person.objects.get(id=2)
nic.name = "Николай Петров"
nic.save()
```

Здесь мы сначала в переменную `nic` прочитали из БД все данные о человеке, информация о котором хранится в БД в записи с `id=2`. Затем во второй строке изменили содержимое поля `name`. И наконец, в третьей строке вернули (записали) всю информацию об этом человеке в БД.

Когда нужно обновить только определенные поля, следует использовать параметр `update_fields`. Так, если, например, в приведенном примере требуется изменить только одно поле `name`, это можно сделать с помощью следующего кода:

```
nic = Person.objects.get(id=2)
nic.name = "Николай Петров"
nic.save(update_fields=["name"])
```

Такой подход позволяет повысить скорость работы приложения, особенно в тех случаях, когда требуется обновить большой массив информации.

Другой способ обновления объектов в БД предоставляет метод `update()` в сочетании с методом `filter()`, которые вместе выполняют один запрос к базе данных. Предположим, что нам нужно обновить имя клиента в записи таблицы БД с `id=2`. Это можно сделать с помощью следующего кода:

```
Person.objects.filter(id=2).update(name="Михаил")
```

При таком подходе не нужно предварительно получать из БД обновляемый объект, что обеспечивает увеличение скорости взаимодействия приложения с БД.

Иногда возникает необходимость изменить значение столбца в БД на основании уже имеющегося значения. В этом случае поможет функция `F()`:

```
from django.db.models import F
Person.objects.all(id=2).update(age = F("age") + 1)
```

Здесь полю `age` присваивается уже имеющееся значение, увеличенное на единицу. При этом важно учитывать, что метод `update` обновляет все записи в таблице, которые соответствуют условию (в приведенном примере таким условием является `id=2`).

Когда необходимо обновить все записи в столбце таблицы БД вне зависимости от условия, то нужно комбинировать метод `update()` с методом `all()` без параметров:

```
from django.db.models import F
Person.objects.all().update(name="Михаил")
Person.objects.all().update(age = F("age") + 1)
```

Здесь всем клиентам будет присвоено значение `Михаил`, а возраст всех клиентов будет увеличен на единицу.

Для обновления записей в БД есть еще один метод — `update_or_create`. Если запись существует, то этот метод ее обновит, а если записи нет, то добавит ее в таблицу:

```
values_for_update={"name": "Михаил", "age": 31}
bob, created = Person.objects.update_or_create(id=2,
                                                defaults = values_for_update)
```

Метод `update_or_create()` здесь принимает два параметра. Первый параметр предоставляет критерий выборки объектов, которые должны обновляться. Второй параметр предоставляет объект со значениями, которые будут переданы записям, соответствующим критерию из первого параметра. Если соответствующих критерию записей обнаружено не будет, в таблицу добавится новый объект, а переменной `created` будет присвоено значение `True`. В приведенном примере критерием для обновления записи является идентификатор записи `id=2`. А поля, которые будут обновлены: `"name": "Михаил"` и `"age": 31`.

### 8.3.4. Удаление данных из БД

Метод `delete()` удаляет информацию из БД. Удаление единственной записи из таблицы можно выполнить с помощью ее `id`. Например:

```
person = Person.objects.get(id=2)
person.delete()
```

Здесь в первой строке в элемент `person` загружена информация из строки таблицы базы данных с `id=2`, а во второй строке вызван метод, который удалил из таблицы БД эту строку.

Если не требуется получение отдельного объекта из базы данных, тогда можно удалить объект одной строкой программного кода с помощью комбинации методов `filter()` и `delete()`:

```
Person.objects.filter(id=4).delete()
```

Эта команда удалит строку с `id=4` непосредственно в базе данных, без предварительной загрузки ее содержимого в приложение.

### 8.3.5. Просмотр строки SQL-запроса к базе данных

Рассмотренные здесь методы при обращении к базе данных фактически используют SQL-запросы, хотя это и скрыто от разработчика. В Django с помощью свойства `query`

можно получить и посмотреть текст выполняемого SQL-запроса. Например, при выполнении кода

```
people = Person.objects.filter(name="Tom").exclude(age=34)
print(people.query)
```

на консоли отобразится следующий SQL-запрос:

```
SELECT "firstapp_person"."id", "firstapp_person"."name",
       "firstapp_person"."age"
  FROM "firstapp_person"
 WHERE ("firstapp_person"."name" = Tom
   AND NOT ("firstapp_person"."age" = 34))
```

## 8.4. Общие принципы взаимодействия форм с моделями данных и шаблонами Django

Форма — это HTML-страница, на которой имеется поле или группа полей. Как уже упоминалось, форма служит для сбора информации от пользователей для последующей отправки на сервер. Формы имеют целый набор виджетов для ввода различных типов данных: текстовые поля, флагки, переключатели, установщики дат и т. д. Формы являются относительно безопасным способом взаимодействия пользовательского клиента и сервера, поскольку они позволяют отправлять данные в POST-запросах, применяя защиту от межсайтовой подделки запроса (Cross Site Request Forgery, CSRF).

В Django основой для создания пользовательских форм является класс `Form`. Примерно так же, как модель Django описывает логическую структуру объекта (фактически таблицы в БД), класс `Form` описывает форму и определяет, как она работает и выглядит.

Подобно тому, как поля класса модели сопоставляются с полями таблицы базы данных, поля класса формы сопоставляются с элементами ввода (`<input>`) HTML-страницы (рис. 8.6).

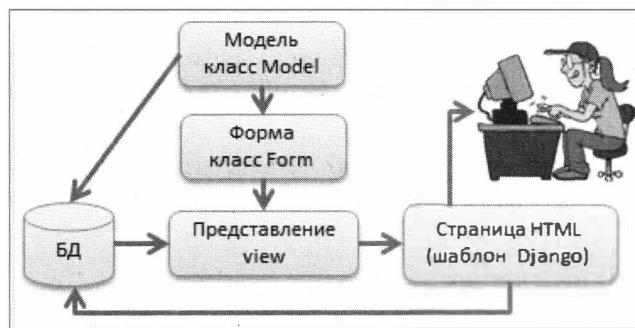


Рис. 8.6. Взаимодействие пользователя с БД через элементы Django

Пользователь на своем компьютере видит только HTML-страницу, но все, что отображается на ней, обеспечивают классы `Model` и `Form`. На первом этапе программист создает модель, в которой описывает структуру таблицы БД. На основе этой модели в БД формируется таблица, которая содержит набор полей. На основе этой же модели создается

форма, которая содержит тот же набор полей. Когда форма через представление (view) отображается на HTML-странице, пользователь опять видит тот же набор полей, поскольку форма в виде объекта передается в шаблон Django. После того, как пользователь ввел данные в эти поля и отправил их на сервер, они записываются в ту таблицу БД, которая определена в модели. В итоге пользователь видит на HTML-странице обновленные данные, которые пришли из БД через представление в ту же форму (или в другую форму).

Поля формы сами по себе являются объектами классов, которые управляют данными формы и проверяют их при отправке в БД. Поля имеют разные типы: дата (`DateField`), файлы (`FileField`), изображения (`ImageField`) и т. п. Зная тип поля из модели, Django самостоятельно определяет, какие проверки необходимо выполнить, какие SQL-запросы потребуются и как отобразить поле пользователю.

Поле формы представляется пользователю в браузере через шаблоны Django как «виджет» HTML. Каждый тип поля имеет соответствующий класс виджета по умолчанию, но при необходимости их можно переопределить.

При рендеринге (визуализации) объекта в Django обычно выполняются следующие шаги:

- формирование данных в представлении (например, получение из БД);
- передача данных в контекст шаблона;
- расширение шаблона с использованием разметки HTML и показа данных в шаблоне.

Визуализация формы в шаблоне включает почти те же шаги, что и визуализация любого другого объекта, но есть некоторые ключевые отличия.

В случае экземпляра модели, которая не содержит данных, нет необходимости что-либо делать с ней в шаблоне. С другой стороны, имеет смысл визуализировать незаполненную форму. Это нужно, когда пользователь должен ввести какие-либо данные. С другой стороны, в форме необходимо показать те данные, которые он уже передал в БД. Поэтому, когда создается форма, она может остаться пустой или предварительно заполненной данными из различных источников, например:

- данными, введенными ранее пользователем в этой форме;
- данными, полученными из других моделей и форм;
- данными, введенными пользователем в БД в текущем сеансе работы с формой.

Последний из этих вариантов наиболее интересен, потому что в этом случае пользователь может не только вводить данные в БД через форму, но и тут же контролировать как корректность введенных данных, так и то, что данные действительно были записаны в БД.

### 8.4.1. Создание форм на основе классов *Form* и *ModelForm*

Проектирование формы начинается с создания пользовательского класса в модуле `forms.py`. Программный код может иметь следующую структуру:

```
from django import forms
class NameForm(forms.Form):
    field_1 = forms.CharField(label='<метка поля>', max_length=100, ...)
    field_2 = forms.CharField(label='<метка поля>', max_length=100, ...)
```

Здесь был выполнен импорт пакета Django `forms`, который отвечает за работу с формами. Затем создана пользовательская форма в виде класса с именем `NameForm` на основе базового класса (`forms.Form`). После этого созданы поля (объекты), которые будут отображены в форме (`field_1`, `field_2`, ...). Все эти поля будут отображены в форме в том порядке, в каком они были созданы.

Создание класса формы `NameForm` при помощи описанного примера, является довольно гибким способом, позволяющим создавать формы произвольной структуры и в связке с любой моделью или моделями.

Однако если вам нужна форма для отображения полей из одной модели, то можно поступить проще. Ведь в самой модели уже содержится большая часть информации, которая необходима для построения формы: сами поля, текстовые метки полей, содержание полей по умолчанию и т. п. Для того чтобы не дублировать информацию в форме, которая уже есть в модели, проще воспользоваться классом `ModelForm`, который помогает создавать формы непосредственно из модели. Класс `ModelForm` может применяться в ваших представлениях (`view`) точно так же, как и "классический" класс формы `Form`.

Базовая реализация `ModelForm` будет содержать те же поля, что и класс формы `Form`. Все что необходимо сделать, — внутри вашего нового класса добавить класс `Meta` и связать его с моделью `model`, а затем перечислить поля модели, которые должны быть включены в форму, в переменной (объекте) `fields`. Можно включить все поля при помощи инструкции `fields = '__all__'` или можно воспользоваться атрибутом `exclude` (вместо `fields`), чтобы определить поля модели, которые нужно исключить. Программный код создания модели на основе класса `ModelForm` будет иметь следующую структуру:

```
from django import forms
from .models import <name_model>
class NameForm (forms.ModelForm):
    class Meta:
        model = <name_model>
        fields = '__all__'
        # fields = ['field_1', 'field_1']
```

Здесь сначала был импортирован пакет Django `forms`, который отвечает за работу с формами. Кроме того, выполнен импорт модели с именем `<name_model>`, которая была создана ранее. Затем выполнена инициализация пользовательской формы в виде класса с именем `NameForm` на основе базового класса (`forms.ModelForm`). После этого создан объект `model` (можно задавать любое имя) на основе импортированной модели. И в заключение создается объект `fields`, который включает все поля из импортированной модели. Можно, конечно, задать имена полей и вручную, как это сделано в закомментированной строке, однако это не рекомендуется, т. к. опять происходит дублирование кода. Явно задавать поля рекомендуется в тех случаях, когда в форме пользователь должен внести данные только в часть полей модели.

## 8.4.2. Связывание форм с представлениями (`view`)

Теперь нужно создать представление (`view`). Рекомендуется в одном представлении и публиковать форму, и обрабатывать данные, которые получены от пользователя через форму. Это позволяет использовать одну и ту же функцию или класс, что сокращает

программный код. Чтобы обработать форму, нужно создать ее экземпляр в представлении и указать URL-адрес, по которому она должна быть опубликована.

Проектирование представления (*view*) начинается с создания пользовательского класса, который создается в модуле `views.py`. Программный код может иметь следующую структуру:

```
from django.http import HttpResponseRedirect
from django.shortcuts import render
from .forms import NameForm
def <get_name_view>(request):
    # если это POST-запрос, то нужно обработать данные, полученные из формы
    if request.method == 'POST':
        # создание экземпляра формы и заполнение ее введенными данными
        form = NameForm(request.POST)
        # проверка валидности формы
        if form.is_valid():
            # обработка данных формы (например, запись в БД)
            form.save()
            # Перенаправление на новый URL:
            return HttpResponseRedirect('new_form')

    # Создание пустой формы
    form = NameForm()
    context = {'form': form}
    return render(request, 'name.html', context)
```

Если представление получает запрос GET, то будет создан пустой экземпляр формы (`form`), который через параметр `context` будет передан в шаблон '`name.html`' для отображения. Этого мы можем ожидать при первом посещении формы через ее URL-адрес.

Если форма отправляется через запрос POST (пользователь ввел данные), представление снова создаст экземпляр формы и заполнит его данными из запроса: это называется «привязкой данных к форме» (теперь это *связанная форма*):

```
form = NameForm(request.POST)
```

После этого вызывается метод формы `is_valid()`. Если в заполненных данных есть ошибки (`form.is_valid()=False`), то происходит возврат к шаблону с формой. На этот раз форма не пустая, она будет заполнена данными, отправленными пользователем. Теперь эти данные можно исправлять и снова отправить на сайт.

Если `form.is_valid()=True` (в данных нет явных ошибок), мы можем получить все проверенные данные формы в атрибуте `cleaned_data` и выполнить с ними любые действия. А можем сохранить все введенные данные в БД простой инструкцией — `form.save()`. После этого в инструкции `return` мы указываем, на какую страницу сайта идти дальше.

### 8.4.3. Связывание представлений (*view*) с шаблонами форм

Осталось создать шаблон Django, который отобразит нашу форму на HTML-странице. В данном примере шаблон имеет имя `<name.html>`.

Самый простой код шаблона будет выглядеть следующим образом:

```
<form action="/your-name/" method="post">
    {%- csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

Все поля формы и их атрибуты будут распакованы на HTML-странице в теге Django {{ form }}.

Теперь рассмотрим организацию связей между таблицами в БД через модели данных.

## 8.5. Организация связей между таблицами в БД через модели данных

Большинство таблиц в базе данных имеют связи между собой. Django позволяет определить три наиболее употребительных типа отношений: «один ко многим», «многие ко многим» и «один к одному».

### 8.5.1. Организация связей между таблицами «один ко многим»

Рассмотрим организацию связи между таблицами БД «один ко многим», при которой одна главная сущность может быть связана с несколькими зависимыми сущностями. Приведем несколько примеров таких связей:

- одна компания, выпускающая множество видов товаров;
- один автомобиль, состоящий из множества составных частей;
- одна гостиница с множеством комнат с разными характеристиками;
- одна книга, у которой несколько авторов;
- один город с множеством улиц;
- одна улица с множеством домов, и т. п.

Покажем, как можно связать две таблицы в БД через связанные модели на примере «одна компания — множество товаров»:

```
from django.db import models
class Company(models.Model):
    name = models.CharField(max_length=30)
class Product(models.Model):
    company = models.ForeignKey(Company, on_delete = models.CASCADE)
    name = models.CharField(max_length=30)
    price = models.IntegerField()
```

В этом примере модель Company представляет собой производителя продукции и является *главной моделью* (главной таблицей в БД), а модель Product представляет собой различные товары, производимые этой компанией, и является *зависимой моделью* (зависимой таблицей в БД).

Конструктор типа `models.ForeignKey` (внешний ключ) в классе `Product` настраивает связь с главной сущностью. Здесь первый параметр указывает, с какой моделью будет создаваться связь, — в нашем случае это модель `Company`. Второй параметр (`on_delete`) задает опцию удаления объекта текущей модели при удалении связанного объекта главной модели. В частности, в приведенном коде задано *каскадное удаление* (`models.CASCADE`). Если из БД будет удалена компания, то автоматически из БД будет удалена и вся продукция, которая выпускается этой компанией.

Параметр `on_delete` может принимать следующие значения:

- `models.CASCADE` — автоматически удаляет строку (строки) из зависимой таблицы, если удаляется связанная строка из главной таблицы;
- `models.PROTECT` — блокирует удаление строки из главной таблицы, если с ней связаны какие-либо строки в зависимой таблице;
- `models.SET_NULL` — устанавливает значение `NULL` при удалении связанной строки из главной таблицы;
- `models.SET_DEFAULT` — устанавливает значение по умолчанию для внешнего ключа в зависимой таблице (в таком случае для этого столбца должно быть задано значение по умолчанию);
- `models.DO_NOTHING` — при удалении связанной строки из главной таблицы не выполняются никакие действия в зависимой таблице.

Итак, внесем изменения в файл `hello/firstapp/models.py`, добавим в него два класса: компания — `class Company` и продукция компании — `class Product` (листинг 8.2, изменения выделены серым фоном).

#### Листинг 8.2. Файл `models.py`

```
from django.db import models
class Person(models.Model):
    name = models.CharField(max_length=20)
    age = models.IntegerField()
    object_person = models.Manager()
    DoesNotExist = models.Manager()
class Company(models.Model):
    name = models.CharField(max_length=30)
class Product(models.Model):
    company = models.ForeignKey(Company, on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    price = models.IntegerField()
```

Теперь выполним миграцию, запустим в окне терминала PyCharm следующую команду (рис. 8.7):

```
python manage.py makemigrations
```

В результате выполнения этой команды на основе моделей `Company` и `Product` в каталоге `migrations` автоматически будет создан новый файл `0002_company_product.py` со следующим содержанием:

```
Terminal: Local +  
System check identified some issues:  
firstapp\migrations\0002_company_alter_person_managers_product.py  
- Create model Company  
- Change managers on person  
- Create model Product
```

Рис. 8.7. Создание миграции для моделей данных Company и Product

```
from django.db import migrations, models
import django.db.models.deletion
import django.db.models.manager
class Migration(migrations.Migration):
    dependencies = [
        ('firstapp', '0001_initial'),
    ]
    operations = [
        migrations.CreateModel(
            name='Company',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
                    serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=30)),
            ],
        ),
        migrations.AlterModelManagers(
            name='person',
            managers=[

                ('object_person', django.db.models.manager.Manager()),
            ],
        ),
        migrations.CreateModel(
            name='Product',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True,
                    serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=30)),
                ('price', models.IntegerField()),
                ('company', models.ForeignKey
                    (on_delete=django.db.models.deletion.CASCADE,
                     to='firstapp.company')),
            ],
        ),
    ]
]
```

Разберемся, какие действия прописаны в этом файле. Во-первых, его номер 0002 говорит о том, что это уже вторая миграция. Во-вторых, что эта миграция является зависимой от файла с первичной миграцией:

```
dependencies = [('firstapp', '0001_initial'),]
```

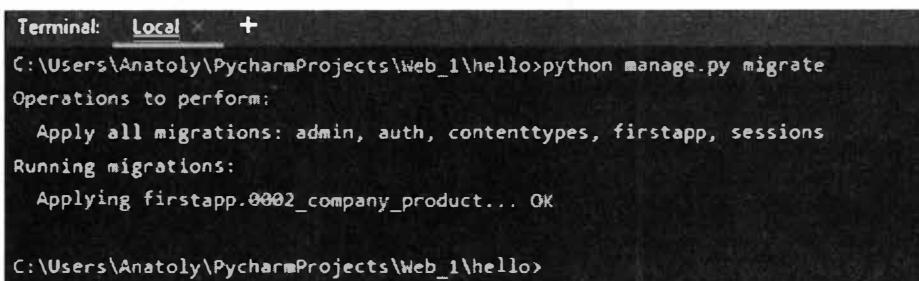
А далее следует код для создания двух моделей данных:

- модели таблицы данных о компании — с именем `name='Company'` и двумя полями: `'id'` и `'name'`;
- модели таблицы данных о продукции компании — с именем `name='Product'` и четырьмя полями: `'id'`, `'name'`, `'price'` и `'company'`.

При этом указано, что в таблице со сведениями о продуктах компании поле `company` является связующим с главной таблицей, и предусмотрено каскадное удаление всех записей для случая, если компания будет удалена из главной таблицы.

Теперь внесем эти изменения в саму базу данных, для чего в окне терминала PyCharm (рис. 8.8) выполним команду:

```
python manage.py migrate
```



```
Terminal: Local +  
C:\Users\Anatoly\PycharmProjects\Web_1\hello>python manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, firstapp, sessions  
Running migrations:  
  Applying firstapp.0002_company_product... OK  
  
C:\Users\Anatoly\PycharmProjects\Web_1\hello>
```

Рис. 8.8. Создание таблиц в БД на основе миграции моделей данных Company и Product

В результате миграции на основе моделей Company и Product в базе данных SQLite автоматически будут созданы таблицы: `firstapp_company` и `firstapp_product` (рис. 8.9).

Как видно из данного рисунка, в сформированных таблицах автоматически созданы ключевые поля для идентификации записей (`id`), а также в таблице `firstapp_product` появилось поле `company_id` для связи этой дочерней таблицы с родительской таблицей `Company`. Именно через это поле в программном коде можно будет получать связанные данные. Например, если требуется получить идентификатор компании, которая производит этот продукт, нужно воспользоваться командой:

```
id_company = Product.objects.get(id=1).company.id
```

Если требуется получить название компании, которая производит этот продукт, нужно воспользоваться командой:

```
name_company = Product.objects.get(id=1).company.name
```

Например, если требуется получить перечень товаров, которые производятся компанией «Мираторг», следует воспользоваться командой:

```
Product.objects.filter(company_name="Мираторг")
```

Здесь нужно обратить особое внимание на выражение `company_name`. С помощью выражения `model__свойство` (обратите внимание: два подчеркивания!) можно использовать свойство главной модели для фильтрации объектов (записей в таблице БД) зависимой модели.

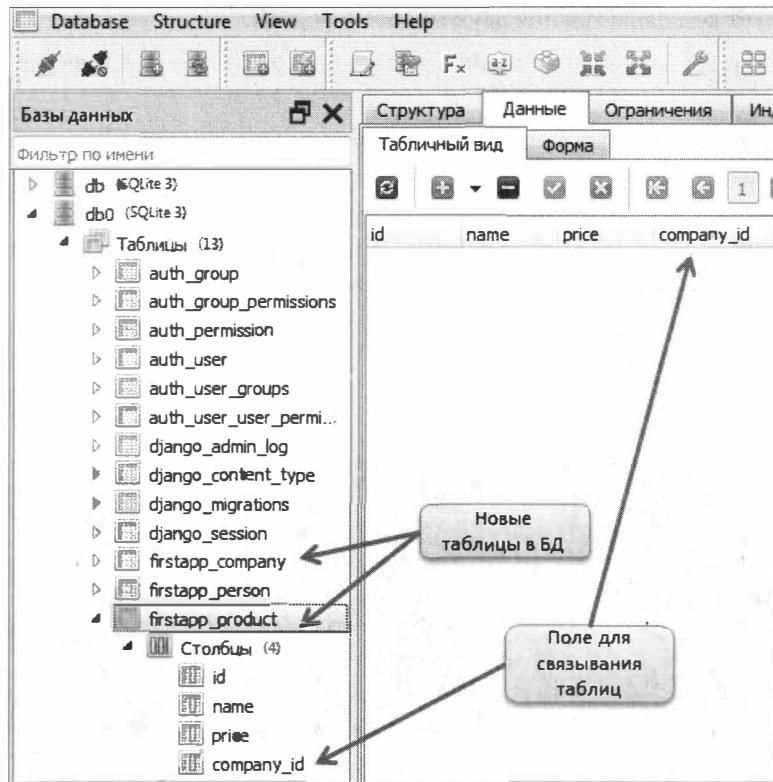


Рис. 8.9. Таблицы, созданные в БД на основе миграции моделей данных Company и Product

С точки зрения модели `Company` (родительская таблица в БД) она не имеет никаких свойств, которые связывали бы ее с моделью `Product` (дочерняя таблица в БД). Но с помощью команды, имеющей следующий синтаксис:

```
"главная_модель". "зависимая_модель" _set
```

можно изменить направление связи. Иными словами, на основании записи из главной модели вы сможете получать связанные записи из зависимой (подчиненной) модели.

Рассмотрим следующий программный код:

```
from .models import Company, Product
firma = Company.objects.get(name="Электрон")
# получение всех товаров фирмы "Электрон"
tovar = firma.product_set.all()
# получение количества товаров фирмы "Электрон"
kol_tovar = firma.product_set.count()
# получение товаров, название которых начинается на "Ноутбук"
Tovar = firma.product_set.filter(name_startwith="Ноутбук")
```

Здесь в объект `firma` из класса `Company` (а фактически из таблицы БД `Company`) мы получаем сведения о компании с именем «Электрон». Затем в переменную `tovar` считываем сведения обо всех продуктах фирмы «Электрон» (по сути, из таблицы БД `firstapp_product`).

Причем с помощью выражения `_set` можно выполнять операции добавления, изменения, удаления объектов зависимой модели из главной модели. Рассмотрим программный код, приведенный в листинге 8.3.

**Листинг 8.3. Пример кода выполнения операций в зависимой модели из главной модели**

```
# на основе класса Company создаем объект firma с именем Электрон
firma = Company.objects.create(name="Электрон")
# создание товара компании
firma.product_set.create(name="Samsung S20", price=42000)
# создание продукта с последующим добавлением его в БД
ipad = Product(name="iPad", price=34200)
# при добавлении необходимо указать параметр bulk =False
firma.product_set.add(ipad, bulk =False)
# исключает из компании все товары,
# при этом товары остаются в БД и не привязаны к компании
# работает, если в зависимой модели ForeignKey(Company, null = True)
# firma.product_set.clear()
# то же самое, только в отношении одного объекта
# ipad = Product.objects.get(name="iPad")
# firma.product_set.remove(ipad)
```

Отметим три метода, которые можно сочетать с выражением `_set`:

- `add()` — добавляет как саму запись в дочернюю таблицу, так и связь между объектом зависимой модели и объектом главной модели. По своей сути метод `add()` вызывает для модели еще и метод `update()` для добавления связи. Однако требуется, чтобы обе модели уже были в базе данных. И здесь применяется параметр `bulk=False`, чтобы объект зависимой модели сразу был добавлен в БД и для него была установлена связь;
- `clear()` — удаляет связь между всеми объектами зависимой модели и объектом главной модели. При этом сами объекты зависимой модели (дочерней таблицы) остаются в базе данных и для их внешнего ключа устанавливается значение `NULL`. Поэтому метод `clear()` будет работать, если в самой зависимой модели при установке связи задан параметр `null=True`, т. е. `ForeignKey(Company, null = True)`;
- `remove()` — так же как и `clear()`, удаляет связь, только между одним объектом зависимой модели и объектом главной модели. При этом все объекты дочерней таблицы остаются в БД. В зависимой модели при установке связи также должен быть указан параметр `null=True`.

## 8.5.2. Организация связей между таблицами «многие ко многим»

Связь «многие ко многим» — это связь, при которой множественным записям из одной таблицы (*A*) могут соответствовать множественные записи из другой таблицы (*B*). Примером такой связи может служить учебное заведение, где преподаватели обучают учащихся. В большинстве учебных заведений (школа, университет) каждый преподаватель обучает многих учащихся, а каждый учащийся может обучаться у нескольких пре-

подавателей. Еще один пример — это книги и авторы книг. У одной книги может быть несколько авторов, в то же время у одного автора может быть несколько книг.

Связь «многие ко многим» создается с помощью трех таблиц: две из них (*A* и *B*) — «источники» и одна таблица — соединительная. Первичный ключ соединительной таблицы (*A-B*) — составной. Он состоит из двух полей: двух внешних ключей, которые ссылаются на первичные ключи таблиц *A* и *B*. Все первичные ключи должны быть уникальными. Это подразумевает, что комбинация полей *A* и *B* должна быть уникальной в таблице *A-B*.

Еще один пример такой связи — номера гостиницы и ее гости. Между таблицами «Гости» и «Комнаты» (или номера) существует связь «многие ко многим»: одну комнату могут заказать многие гости в течение определенного времени, и в течение этого же промежутка времени гость может заказать в гостинице разные комнаты. Однако соединительная таблица в таком случае может не являться классической, состоящей только из двух внешних ключей. Она может быть отдельной сущностью с дополнительными полями, имеющей связи с двумя другими сущностями (при этом уникальность ключей должна соблюдаться).

Вследствие природы отношения «многие ко многим» совершенно неважно, какая из таблиц родительская, а какая — дочерняя, потому что по своей сути такой тип отношений является симметричным.

Для представления отношения «многие ко многим» Django самостоятельно создает промежуточную связывающую таблицу. По умолчанию имя этой таблицы образуется из имен двух соединяемых таблиц.

Рассмотрим, как можно связать две таблицы в БД через связанные модели на примере: «много учебных курсов — много студентов». Для создания отношения «многие ко многим» применяется тип связи `ManyToManyField`. Итак, добавим в файл `hello/firstapp/models.py` код листинга 8.4.

#### Листинг 8.4. Измененный код файла `models.py`

```
from django.db import models
class Course(models.Model):
    name = models.CharField(max_length=30)
class Student(models.Model):
    name = models.CharField(max_length=30)
    courses = models.ManyToManyField(Course)
```

В этом примере модель `Course` представляет собой учебные курсы, а модель `Student` — студентов. Здесь нет родительской и дочерней таблицы, они равнозначны.

Новая сущность `courses`, устанавливающая отношение «многие ко многим», создается с помощью конструктора `models.ManyToManyField`. В результате генерируется промежуточная таблица, через которую, собственно, и будет осуществляться связь.

Теперь запустим в окне терминала PyCharm (рис. 8.10) миграцию, подав команду

```
python manage.py makemigrations
```

В результате выполнения этой команды на основе моделей `Course` и `Student` в каталоге `migrations` автоматически будет создан новый файл `0003_course_student.py`, имеющий следующее содержание:

```
Terminal: Local +  
C:\Users\Anatoly\PycharmProjects\Web_1\hello>python manage.py makemigrations  
Migrations for 'firstapp':  
    firstapp\migrations\0003_course_student.py  
        - Create model Course  
        - Create model Student
```

Рис. 8.10. Создание миграции для моделей данных Course и Student

```
operations = [  
    migrations.CreateModel(  
        name='Course',  
        fields=[('id', models.AutoField(auto_created=True,  
                                         primary_key=True, serialize=False,  
                                         verbose_name='ID')),  
               ('name', models.CharField(max_length=30)),  
        ],  
    ),  
    migrations.CreateModel(  
        name='Student',  
        fields=[('id', models.AutoField(auto_created=True,  
                                         primary_key=True, serialize=False,  
                                         verbose_name='ID')),  
               ('name', models.CharField(max_length=30)),  
               ('courses',  
                models.ManyToManyField  
(to='firstapp.Course')),  
        ],  
    ),  
]
```

Здесь прописан код для создания двух моделей данных:

- модели таблицы данных о курсах — с именем `name='Course'` и двумя полями: `'id'` и `'name'`;
- модели таблицы данных о студентах — с именем `name='Student'` и двумя полями: `'id'` и `'name'`.

При этом указано, что между таблицами имеется связующая таблица, которая обеспечивает связь «многие ко многим», и к этим связям можно обращаться через свойство `courses`.

Теперь внесем эти изменения в саму базу данных, для чего в окне терминала PyCharm (рис. 8.11) выполним команду `python manage.py migrate`.

В результате миграции на основе моделей `Course` и `Student` в базе данных SQLite автоматически будут созданы уже не две, а три таблицы: `firstapp_course`, `firstapp_student` и `firstapp_course_student` (рис. 8.12).

В нашем случае таблица с именем `firstapp_student_courses` выступает в качестве связующей таблицы. Ей автоматически присвоено имя по шаблону:

`имя приложения+имя_таблицы+имя_связующего поля из таблицы`

```
Terminal: Local × +
C:\Users\Anatoly\PycharmProjects\Web_1\hello>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, firstapp, sessions
Running migrations:
  Applying firstapp.0003_course_student... OK
```

Рис. 8.11. Создание таблиц в БД на основе миграции моделей данных Course и Student

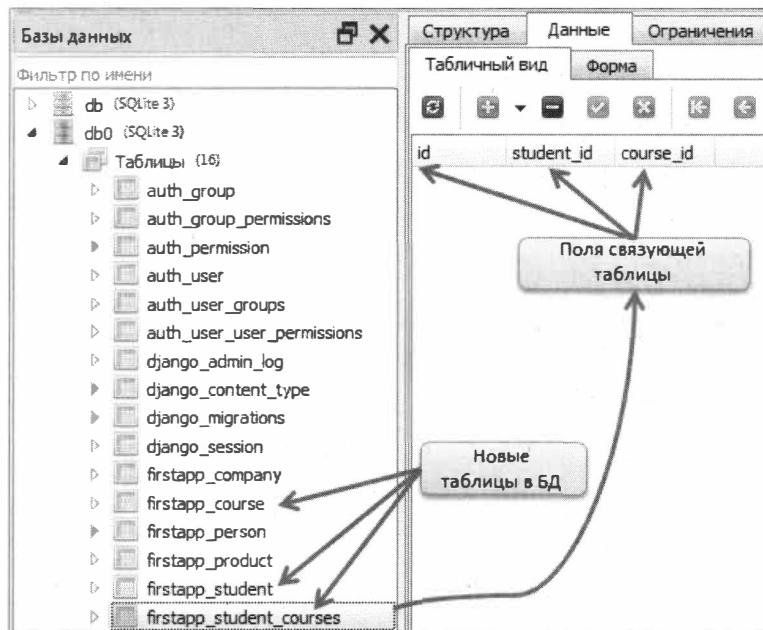


Рис. 8.12. Три таблицы, созданные в БД на основе миграции моделей данных с множественными связями Course и Student

А в самой связующей таблице имеются всего два поля с ключами из двух связанных таблиц: `student_id` и `course_id`.

Через свойство `courses` в модели `Student` мы можем получать курсы, связанные со студентом, и управлять ими. Рассмотрим следующий код:

```
# создадим студента с именем Виктор
stud_viktor = Student.objects.create(name="Виктор")
# создадим один курс и добавим в него Виктора
stud_viktor.courses.create(name="Математика")
# получим все курсы студента Виктора
all_courses = Student.objects.get(name="Виктор").courses.all()
# получаем всех студентов, которые посещают курс Математика
all_student = Student.objects.filter(courses__name="Математика")
```

Стоит обратить внимание на последнюю строку кода, где выполняется выборка студентов по посещаемому курсу. Для передачи в метод `filter()` имени курса задан параметр

с именем, начинающимся с названия свойства, через которое идет связь со второй моделью (`courses`). И далее через два знака подчеркивания указывается имя свойства второй модели, например `courses__name` или `courses__id`. Иными словами, мы можем получить информацию о курсах студента через свойство `courses`, которое определено в модели `Student`.

Однако имеется возможность получать информацию и о студентах, которые изучают определенные курсы. В этом случае нужен синтаксис `_set`. Рассмотрим следующий программный код:

```
# создадим курс программирования на Python
kurs_python = Course.objects.create(name="Python")
# создаем студента и добавляем его на курс
kurs_python.student_set.create(name="Виктор")
# отдельно создаем студента и добавляем его на курс
alex = Student(name="Александр")
alex.save()
kurs_python.student_set.add(alex)
# получим всех студентов курса
students = kurs_python.student_set.all()
# получим количество студентов по курсу
number = kurs_python.student_set.count()
# удаляем с курса одного студента
kurs_python.student_set.remove(alex)
# удаляем всех студентов с курса
kurs_python.student_set.clear()
```

Следует учитывать, что не всегда такая организация связи «многие ко многим» может подойти. Например, в нашем случае создается промежуточная таблица, которая хранит только `id` студента и `id` курса. Если нам нужно в промежуточной таблице хранить еще какие-либо данные, например дату зачисления студента на курс, его оценки и т. д., то такая конфигурация не подойдет. И тогда правильнее будет создать промежуточную сущность вручную (например, запрос или хранимую процедуру), которая связана отношением «один ко многим» с обеими моделями.

### 8.5.3. Организация связей между таблицами «один к одному»

При организации связи «один к одному» каждая запись из таблицы *A* может быть ассоциирована только с одной записью таблицы *B*. Связь «один к одному» легко моделируется в одной таблице. Записи такой таблицы содержат данные, которые находятся в связи «один к одному» с первичным ключом.

В редких случаях связь «один к одному» моделируется с использованием двух таблиц. Такой вариант иногда необходим, чтобы преодолеть ограничения СУБД или с целью увеличения производительности (производится, например, вынесение ключевого поля в отдельную таблицу для ускорения поиска по другой таблице). Или вы сами захотите разнести две сущности, имеющие связь «один к одному», по разным таблицам. Например, всю базовую информацию о пользователе (имя, возраст, электронный адрес и пр.) выделить в одну модель, а его учетные данные (логин, пароль, время последнего входа

в систему, количество неудачных входов и т. п.) — в другую модель. Но обычно наличие двух таблиц в связи «один к одному» считается плохой практикой.

Рассмотрим, как можно связать две таблицы в БД через связанные модели на примере: «пользователь системы — учетные данные пользователя». Для создания отношения «один к одному» применяется тип связи `models.OneToOneField()`. Добавим в файл `models.py` код листинга 8.5.

### Листинг 8.5. Измененный код файла models.py

```
from django.db import models
class User(models.Model):
    name = models.CharField(max_length=20)
class Account(models.Model):
    login = models.CharField(max_length=20)
    password = models.CharField(max_length=20)
    user = models.OneToOneField(User,
                                on_delete = models.CASCADE,
                                primary_key = True)
```

Здесь мы создали модель пользователя (`User`) с одним полем `name` и модель учетных данных пользователя (`Account`) с двумя полями: `login` и `password`. Для создания отношения «один к одному» был применен конструктор типа `models.OneToOneField()`. Его первый параметр указывает, с какой моделью будет ассоциирована эта сущность (в нашем случае ассоциация с моделью `User`). Второй его параметр (`on_delete=models.CASCADE`) говорит, что данные текущей модели (`Account`) будут удаляться при удалении связанного объекта главной модели (`User`). Третий параметр (`primary_key=True`) указывает, что внешний ключ (через который идет связь с главной моделью) одновременно будет выступать и в роли первичного ключа и соответственно создавать отдельное поле для первичного ключа.

Теперь внесем эти изменения в саму базу данных, для чего выполним в окне терминала PyCharm последовательно две команды:

```
python manage.py makemigrations  
python manage.py migrate
```

В результате миграции в базе данных SQLite будут созданы следующие таблицы: таблица `firstapp_user` с полями `id` и `name` и таблица `firstapp_account` с полями `login`, `password` и `user_id` (рис. 8.13).

Как можно видеть, в таблице `firstapp_account` нет собственного первичного ключа (`id`) — его роль выполняет ключ `user_id`, который одновременно служит для связи с таблицей `firstapp_user`.

С помощью свойства `users` в модели `Account` мы можем манипулировать связанным объектом модели `User`:

```
# изменяем имя пользователя
acc.user.name = "Саша"
# сохраняем изменения в БД
acc.user.save()
```

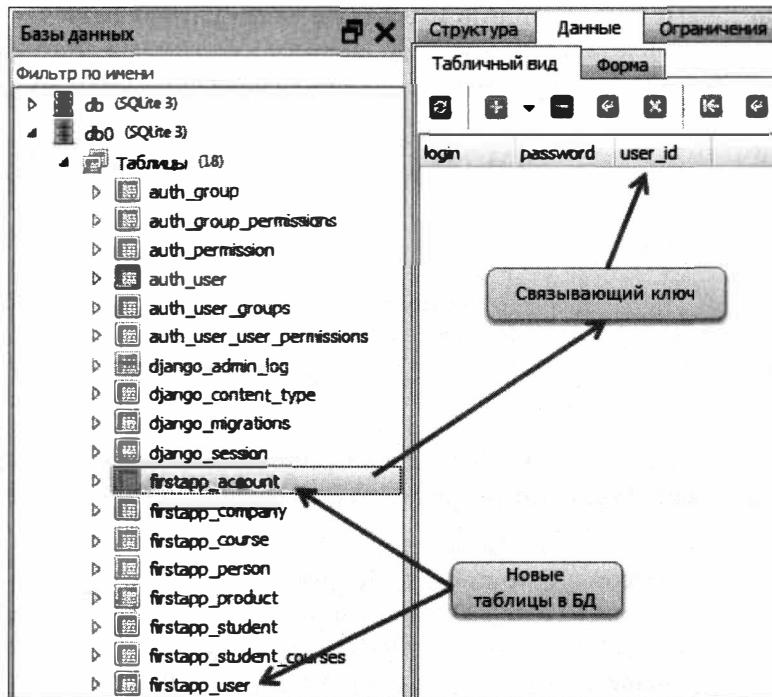


Рис. 8.13. Таблицы, созданные в БД на основе миграции моделей данных со связью «один к одному» на основе моделей User и Account

При этом через модель User мы также можем влиять на связанный объект Account. Несмотря на то что явным образом в модели User определено только одно свойство — name, при связи «один к одному» неявно создается еще одно свойство, которое называется по имени зависимой модели и указывает на связанный объект этой модели. В нашем случае это свойство будет называться account:

```
# создадим пользователя Александра
alex = User.objects.create(name="Александр")
# создадим аккаунт пользователя
acc = Account(login = "1234", password="6565")
alex.account = acc
alex.account.save()
# обновляем данные
alex.account.login = "qwerty"
alex.account.password = "123456"
alex.account.save()
```

Теперь рассмотрим совместную работу форм и моделей данных на примерах.

## 8.6. Пример работы с объектами модели данных (чтение и запись информации в БД)

Рассмотрим работу с моделями и формами на простом примере. Для этого воспользуемся моделью данных Person, которую мы создали в предыдущих разделах. Эта модель данных хранится в файле hello/firstapp/models.py. Немного изменим код этого модуля с учетом тех сведений, которые получены из предыдущего раздела (листинг 8.6).

### Листинг 8.6. Измененный код модуля models.py

```
from django.db import models
class Person(models.Model):
    name = models.CharField(max_length=20,
                           verbose_name="Имя клиента")
    age = models.IntegerField(verbose_name="Возраст клиента")
    object_person = models.Manager()
    DoesNotExist = models.Manager()
```

Согласно приведенной модели в БД предусмотрены два поля для хранения сведений о клиенте: имя клиента (`name`) и возраст клиента (`age`).

Здесь `models.Manager()` — это интерфейс, через который для моделей Django предоставляются операции запросов к базе данных. По крайней мере, один Manager существует для каждой модели в приложении Django. По умолчанию Django добавляет Manager с именем `objects` в каждый класс модели Django. Однако если вы хотите использовать имя для Manager, отличное от `objects`, то можете переименовать его для каждой модели. Чтобы переименовать Manager для данного класса, можно воспользоваться, например, следующей инструкцией:

```
object_person = models.Manager()
```

Здесь мы переименовали менеджер и дали ему более осмысленное имя, ассоциированное со сведениями о клиентах, хранящихся в БД, — `object_person`.

Атрибут `DoesNotExist` позволяет обрабатывать ошибки в тех случаях, когда ожидаемый объект по каким-то причинам не найден. Нам он пригодится в модуле удаления записи из БД.

В предыдущем разделе при выполнении миграции в базе данных (в файле `db.sqlite3`) была создана таблица для хранения сведений о клиентах с именем `firstapp_person` и тремя полями: `id` — идентификатор записи, `name` — имя клиента и `age` — возраст клиента. Именно в этой таблице и будут храниться данные о клиентах.

Как уже упоминалось, в Django за взаимодействие с БД отвечает представление (`view`). На следующем шаге мы в файле `views.py` посредством функции `index()` получим сведения из БД, а функция `my_form()` вызовет форму ввода данных о клиентах.

Модифицированный код модуля `hello/firstapp/views.py` приведен в листинге 8.7 (изменения выделены серым фоном).

**Листинг 8.7. Измененный код модуля views.py**

```
from .forms import UserForm
from django.shortcuts import render, redirect
from .models import Person
def index(request):
    my_text = 'Изучаем модели Django'
    people_kol = Person.objects.all().count()
    context = {'my_text': my_text, "people_kol": people_kol}
    return render(request, "firstapp/index.html", context)
def about(request):
    return render(request, "firstapp/about.html")
def contact(request):
    return render(request, "firstapp/contact.html")
# взаимодействие с формой ввода данных о клиентах
def my_form(request):
    if request.method == "POST": # пользователь отправил данные
        form = UserForm(request.POST) # создание экземпляра формы
        if form.is_valid(): # проверка валидности формы
            form.save() # запись данных в БД
            # остаемся на той же странице, обновляем форму
            # Загрузить форму для ввода клиентов
            my_text = 'Сведения о клиентах'
            people = Person.objects.all()
            form = UserForm()
            context = {'my_text': my_text, "people": people, "form": form}
            return render(request, "firstapp/my_form.html", context)
```

Здесь из моделей импортирован класс Person, а также добавлен импорт пакета redirect:

```
from .models import Person
```

Изменен код функции index(), в которой мы формируем строковую переменную my\_text. Далее создаем объект people\_col и с помощью метода

```
Person.objects.all().count()
```

получаем число записей о клиентах, которые хранятся в БД. Затем формируем словарь context, в который включаем текстовую переменную my\_text и объект people\_count. На последнем шаге через функцию render передаем словарь context на главную страницу сайта (в шаблон Django firstapp/index.html).

Изменен код функции my\_form(). Здесь проверяется условие, не нажал ли пользователь кнопку Отправить в форме my\_form (if request.method == "POST"). Если это условие выполняется, то проверяется валидность формы и данные сохраняются в БД. После этого мы остаемся на той же странице, и форма для ввода данных о клиентах обновляется (происходит перезагрузка формы).

Если клиент не вводил данные, то происходит простая загрузка формы. Здесь формируется текстовая переменная my\_text и создается объект people, который получает все записи из таблицы БД со сведениями о клиентах. После этого создается объект form на основе класса Userform. Все данные, включая саму форму, упаковываются в словарь

context. На последнем шаге вызывается шаблон Django `firstapp/my_form.html`, в который передается словарь context.

Теперь изменим шаблон главной страницы `templates/firstapp/index.html`, в которой будут отображаться данные о числе клиентов, введенных в БД (листинг 8.8).

#### Листинг 8.8. Измененный код файла index.html

```
{% extends "firstapp/base.html" %}
{% block title %}Главная страница{% endblock title %}
{% block header %}Это главная страница сайта - index.html.{% endblock header %}
{% block content%}
    <div class="container-fluid my-2">
        <div class="row">
            <div class="col text-center">
                <div>{{ my_text }}</div>
                {% if people_kol > 0 %}
                    <h5>
                        Количество введенных пользователей - {{ people_kol }}
                    </h5>
                {% endif %}
            </div>
        </div>
    </div>
{% endblock content %}
```

В этом шаблоне использованы теги Django для вывода данных и теги Bootstrap для форматирования страницы. В данном модуле сначала выводится значение переменной `{{my_text}}`. Затем проверяется условие, есть ли в БД записи о клиентах, которые пользователь ввел через форму `my_form`, – `{% if people_kol > 0 %}`. Здесь `people_kol` — это объект, в который из БД было считано число записей о введенных в БД клиентах (передан в шаблон из функции `index`). Если в БД имеются записи о клиентах, то их число будет выведено в теге `{{ people_kol }}`.

Изменим шаблон формы для ввода данных о клиентах, т. е. файл `hello/templates/firstapp/my_form.html` (листинг 8.9).

#### Листинг 8.9. Измененный код шаблона my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}
<div class="container-fluid text-start my-2 border border-5 border-warning">
    <h5>Формы и модели данных</h5>
    <form method="POST">
        {% csrf_token %}
        <div class="form-group my-2">
            {% for field in form %}
                <div>{{field.label_tag}}</div>
                <div>{{field}}</div>
            {% endfor %}
        </div>
    </form>
</div>
```

```
<div class="error">{{ field.errors }}</div>
{%
    endfor
}
</div>
<input type="submit" value="Отправить" >
{%
    if people.count > 0 %
        <h5>Список клиентов</h5>
        <table class="table table-striped table-bordered
                    text-start">
            <thead>
                <tr>
                    <th>№</th>
                    <th>ФИО клиента</th>
                    <th>Возраст</th>
                </tr>
            </thead>
            <tbody>
                {%
                    for person in people %
                }
                <tr>
                    <td>{{ person.id }}</td>
                    <td>{{ person.name }}</td>
                    <td>{{ person.age }}</td>
                </tr>
                {%
                    endfor %
                }
            </tbody>
        </table>
    {%
        endif %
    }
</form>
</div>
{%
    endblock content %
}
```

Теперь в форме присутствуют два тега Django:

- `{% for field in form %}` — организует цикл вывода полей формы для ввода данных;
- `{% if people.count > 0 %}` — отображает сведения о введенных данных.

Первый тег просто отображает поля формы, через которые пользователь вводит сведения о клиенте, и после завершения этого цикла выводит кнопку для отправки данных на сервер.

Во втором теге выполняется проверка, есть ли в БД записи о клиенте. Если в БД пусто, то будет просто загружен шаблон этой страницы без данных. Если в БД есть записи о клиентах, то на основе тега `<table>` будет сформирована таблица, в которую с использованием тега Django для организации цикла `{% for person in people %}` будут выведены все записи из БД со сведениями о клиентах:

- `person.id` — идентификатор записи в БД со сведениями о клиенте;
- `person.name` — имя клиента;
- `person.age` — возраст клиента.

Все элементы данного шаблона помещены в адаптивный контейнер Bootstrap — `container-fluid`, поэтому страница сайта будет одинаково хорошо выглядеть и на большом экране персонального компьютера, и на экране смартфона.

Что ж, у нас все готово, и можно запустить приложение. При первом запуске мы получим страницу, которая будет иметь вид, показанный на рис. 8.14.

Как видно из данного рисунка, на главной странице сайта отсутствует таблица со сведениями о клиентах, т. к. в базе данных еще нет ни одной записи. В главном меню выберем опцию **Форма**, перейдем на страницу с формой для ввода данных о клиентах и введем туда несколько записей (рис. 8.15).



Рис. 8.14. Начальный вид страницы index.html

<b>Формы и модели данных</b>
Имя клиента:
<input type="text" value="Климов Игорь"/>
Возраст клиента:
<input type="text" value="63"/>
<b>Отправить</b>

Рис. 8.15. Ввод сведений о клиенте в форме my\_form.html

Когда пользователь нажмет на кнопку **Отправить**, данные о клиенте будут переданы на сервер и записаны в БД. После этого форма обновится, поля для ввода данных очистятся, и появится таблица с информацией о клиентах, полученная из базы данных (рис. 8.16).

Когда пользователь заведет данные о нескольких клиентах (пользователях), информация о них будет считана из БД и отображена в этой же форме (рис. 8.17).

Это заголовок главной страницы сайта

Главная страница Форма Изображения Файлы О компании Контакты

## Изучаем формы Django

**Формы и модели данных**

Имя клиента:

Возраст клиента:

**Отправить**

**Список клиентов**

№	ФИО клиента	Возраст
1	Климов Игорь	63

Это подвал (footer) страниц сайта

Рис. 8.16. Отображение введенной информации о клиенте в форме my\_form.html

Это заголовок главной страницы сайта

Главная страница Форма Изображения Файлы О компании Контакты

## Изучаем формы Django

**Формы и модели данных**

Имя клиента:

Возраст клиента:

**Отправить**

**Список клиентов**

№	ФИО клиента	Возраст
1	Климов Игорь	63
2	Житомирский Владимир	59
3	Шестов Алексей	55
4	Чурилко Николай	62

Это подвал (footer) страниц сайта

Рис. 8.17. Отображение введенной информации о клиентах, введенных в БД, в форме my\_form.html

Если теперь обратиться к главной странице сайта, то на ней будет отображена информация о числе клиентов, введенных в БД (рис. 8.18).

Если мы теперь с использованием менеджера SQLiteStudio откроем содержимое таблицы firstapp\_person, то увидим в ней данные о клиентах, которые пользователь занес в БД с помощью созданного нами приложения (рис. 8.19).

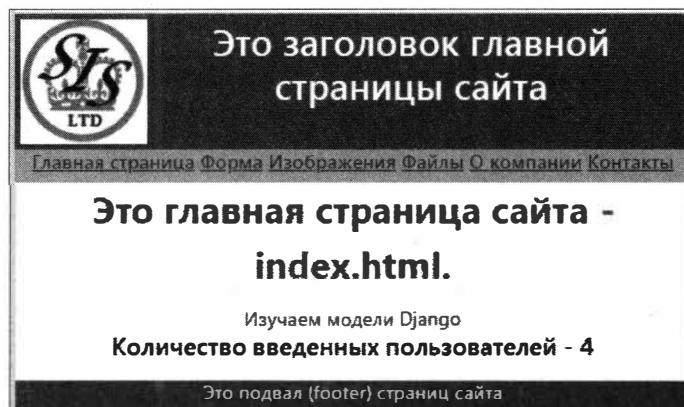


Рис. 8.18. Вид страницы index.html после ввода данных о нескольких пользователях

The screenshot shows the SQLiteStudio application interface. On the left, the 'Базы данных' (Databases) panel lists two databases: 'db (SQLite 3)' and 'db0 (SQLite 3)'. Under 'db0 (SQLite 3)', the 'Таблицы (11)' (Tables) section is expanded, showing tables like auth\_group, auth\_group\_permissions, auth\_permission, auth\_user, auth\_user\_groups, auth\_user\_user\_permissions, django\_admin\_log, django\_content\_type, django\_migrations, django\_session, and firstapp\_person. An arrow points from the 'firstapp\_person' table entry in this list to the main table view on the right. The main table view displays the 'firstapp\_person' table with the following data:

id	name	age
1	Климов Игорь	63
2	Житомирский Владимир	59
3	Шестов Алексей	55
4	Чурилко Николай	62

Рис. 8.19. Информация о пользователях в базе данных SQLite

Как видно из данного рисунка, мы записали информацию в базу данных через интерфейс HTML-страницы. При этом были задействованы два метода: `Person.objects.all()` (получение информации из БД) и `klient.save()` (запись информации о клиентах в БД). Обратите внимание, что в программном коде не был явно использован язык SQL. Хотя фактически именно с его помощью были выполнены операции взаимодействия с БД. Просто это произошло в фоновом режиме благодаря моделям и менеджерам Django.

## 8.7. Пример работы с объектами модели данных: редактирование и удаление информации из БД

Рассмотрим пример с редактированием и удалением объектов модели. Для этого продолжим работу с проектом из предыдущего раздела. Для начала добавим в файл `hello/firstapp/views.py` функции, которые будут выполнять редактирование (`edit_form`) и удаление (`delete`) информации из БД (листинг 8.10).

### Листинг 8.10. Измененный код модуля `views.py`

```
from .forms import UserForm
from django.shortcuts import render, redirect
from .models import Person
from django.http import HttpResponseRedirect
# изменение данных о клиенте в БД
def edit_form(request, id):
    person = Person.objects.get(id=id)
    # Если пользователь вернул отредактированные данные
    if request.method == "POST":
        person.name = request.POST.get("name")
        person.age = request.POST.get("age")
        person.save()
        return redirect('my_form')
    # Если пользователь отправляет данные на редактирование
    data = {"person": person}
    return render(request, "firstapp/edit_form.html", context=data)
# удаление данных о клиенте из БД
def delete(request, id):
    try:
        person = Person.objects.get(id=id)
        person.delete()
        return redirect('my_form')
    except Person.DoesNotExist:
        return HttpResponseRedirect("<h2>Объект не найден</h2>")
```

Функции `index()`, `about()`, `contact()`, `my_form` остаются без изменений.

Функция `edit_form()` выполняет редактирование объекта. Она в качестве параметра принимает идентификатор объекта `id` из базы данных. На основе этого идентификатора мы создаем объект `person` и с помощью метода `Person.objects.get(id=id)` загружаем в него данные о клиенте. После этого программа разделяется на две ветви. Если поступит запрос POST, т. е. пользователь отправил новые (измененные) данные о клиенте, мы сохраняем эти данные в БД и выполняем переадресацию на форму ввода данных. Если поступит запрос GET, отображаем пользователю страницу `edit_form.html` с формой для редактирования объекта (информации о клиенте). Эта страница представляет собой шаблон Django, в который через объект `data` передаются сведения о клиенте (`person`), которые нужно отредактировать.

Функция `delete()` аналогичным образом находит объект, удаляет его и перезагружает страницу ввода данных. Поскольку в случае отсутствия объекта мы можем столкнуться

с исключением Person.DoesNotExist (объект не найден), то нужно обработать это исключение (например, эта запись была удалена другим пользователем). И если объект не будет найден, пользователю вернется сообщение об ошибке 404 — через вызов метода return HttpResponseRedirect().

На следующем шаге нужно создать HTML-страницу, в которой пользователь сможет редактировать данные о клиенте. Добавим новый шаблон в папку hello/templates/firstapp/edit\_form.html со следующим содержимым (листинг 8.11).

#### Листинг 8.11. Код шаблона edit\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}


<h5>Редактирование данных</h5>
<form method="POST">
    {% csrf_token %}
    <p>
        <label>Введите имя</label><br>
        <input type="text" name="name" value="{{person.name}}"/>
    </p>
    <p>
        <label>Введите возраст</label><br>
        <input type="number" name="age" value="{{person.age}}"/>
    </p>
    <input type="submit" value="Сохранить" >
</form>
</div>
{% endblock content %}


```

В этом программном модуле в адаптивном контейнере container-fluid находятся два поля с метками для корректировки данных и кнопка для сохранения в БД скорректированной информации (выделено серым фоном).

Чтобы обеспечить редактирование и удаление объектов (сведений о клиентах) непосредственно в форме ввода данных, изменим шаблон hello/templates/firstapp/my\_form.html, где выводится список объектов (клиентов), добавив в него ссылки на страницы редактирования и удаления объектов из БД (листинг 8.12, изменения выделены серым фоном).

#### Листинг 8.12. Измененный код шаблона my\_form.html

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}

```

```
<h5>Формы и модели данных</h5>
<form method="POST">
    {% csrf_token %}
    <div class="form-group my-2">
        {% for field in form %}
            <div>{{field.label_tag}}</div>
            <div>{{field}}</div>
            <div class="error">{{field.errors}}</div>
        {% endfor %}
    </div>
    <input type="submit" value="Отправить" >
    {% if people.count > 0 %}
        <h5>Список клиентов</h5>
        <table class="table table-striped table-bordered
                    text-start">
            <thead>
                <tr>
                    <th>№</th>
                    <th>ФИО клиента</th>
                    <th>Возраст</th>
                </tr>
            </thead>
            <tbody>
                {% for person in people %}
                    <tr>
                        <td>{{ person.id }}</td>
                        <td>{{ person.name }}</td>
                        <td>{{ person.age }}</td>
                        <td>
                            <a href="edit_form/{{person.id}}">Изменить</a>
                            <a href="delete/{{person.id}}">Удалить</a>
                        </td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% endif %}
    </form>
</div>
{% endblock content %}
```

Здесь в таблицу добавлена еще одна колонка, в которую вставлены две ссылки:

- [Изменить](edit_form/{{person.id}}) — обращение к функции редактирования данных о клиентах (по сути, к форме `edit_form.html`);
- [Удалить](delete/{{person.id}}) — обращение к функции удаления данных о клиентах.

Затем внесем изменения в файл `hello/firstapp/urls.py`, сопоставим функции `edit_form` и `delete` с маршрутами (листинг 8.13, изменения выделены серым фоном).

### Листинг 8.13. Измененный код модуля urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
]
```

Здесь при указании маршрута к функциям `views.edit_form` и `views.delete` в URL-адрес добавлен идентификатор клиента в БД в виде целого числа (`<int:id>`). Поскольку вызов этих функций выполняется из формы `my_form`, то она задана в качестве префикса при указании их URL-адреса (`my_form/edit_form/`, `my_form/delete/`).

Запустим наш локальный сервер:

```
python manage.py runserver
```

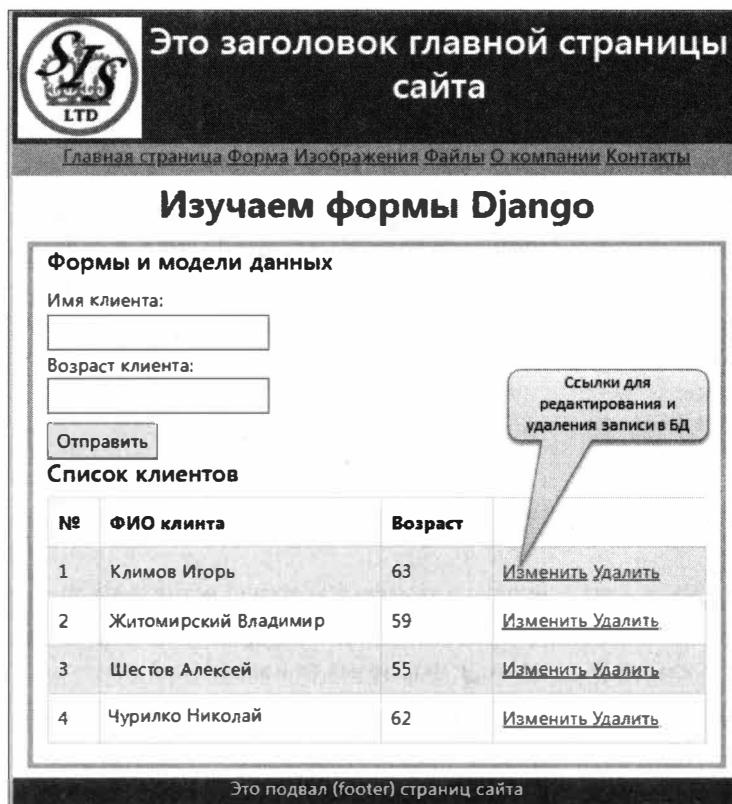


Рис. 8.20. Форма с возможностью редактирования сведений о клиентах

Загрузим форму для ввода данных. Теперь в таблице формы рядом с каждым объектом (клиентом) появится колонка с двумя ссылками: **Изменить** и **Удалить** (рис. 8.20).

При нажатии на ссылку **Изменить** откроется страница для редактирования сведений о клиенте. Изменим возраст одного из них с 63 на 65 лет (рис. 8.21).

После того как будет нажата кнопка **Сохранить**, мы вернемся на главную страницу, где будет видно, что параметр клиента **Возраст** действительно изменился (рис. 8.22).

The screenshot shows a web page titled "Изучаем формы Django". A modal window titled "Редактирование данных" contains fields for "Введите имя" (Klimov Igor) and "Введите возраст" (65). A tooltip "Меняем возраст" points to the age input field. A "Сохранить" button is at the bottom. The footer says "Это подвал (footer) страницы сайта".

Рис. 8.21. Страница для изменения сведений о клиенте

The screenshot shows the main site index.html page with a heading "Это главная страница сайта - index.html." and a subtitle "Список пользователей". A tooltip "Результаты изменения возраста" points to the age column in a table. The table has columns: №, ФИО клиента, Возраст, and two links (Изменить, Удалить). The data is:

№	ФИО клиента	Возраст	
1	Климов Игорь	65	<a href="#">Изменить</a> <a href="#">Удалить</a>
2	Житомирский Владимир	59	<a href="#">Изменить</a> <a href="#">Удалить</a>
3	Шестов Алексей	55	<a href="#">Изменить</a> <a href="#">Удалить</a>
4	Чурилко Николай	62	<a href="#">Изменить</a> <a href="#">Удалить</a>

Это подвал (footer) страницы сайта

Рис. 8.22. Фрагмент главной страницы сайта после редактирования возраста клиента

Проверим, действительно ли в базу данных были внесены эти изменения. Откроем содержимое таблицы `forstapp_person` с помощью SQLiteStudio (рис. 8.23).

Как видно из данного рисунка, в БД действительно изменилось содержимое поля `age`.

Database Structure View Tools Help

Базы данных

Фильтр по имени

- db (SQLite 3)
- dbo (SQLite 3)
  - Таблицы (11)
    - auth\_group
    - auth\_group\_permissions
    - auth\_permission
    - auth\_user
    - auth\_user\_groups
    - auth\_user\_user\_permit...
    - django\_admin\_log
    - django\_content\_type
    - django\_migrations
    - django\_session
    - firstapp\_person

Структура Данные Ограничения Индексы

Табличный вид Форма

	<b>id</b>	<b>name</b>	<b>age</b>
1	1	Климов Игорь	65
2	2	Житомирский Владимир	59
3	3	Шестов Алексей	55
4	4	Чурилко Николай	62

Рис. 8.23. Новое значение возраста клиента в БД после его редактирования на HTML-странице

Это заголовок главной страницы сайта

Логотип LTD Главная страница Форма Изображения Файлы О компании Контакты

## Изучаем формы Django

Формы и модели данных

Имя клиента:

Возраст клиента:

Отправить

Список клиентов

Клиент с id=1 удален

№	ФИО клиента	Возраст	
2	Житомирский Владимир	59	<a href="#">Изменить</a> <a href="#">Удалить</a>
3	Шестов Алексей	55	<a href="#">Изменить</a> <a href="#">Удалить</a>
4	Чурилко Николай	62	<a href="#">Изменить</a> <a href="#">Удалить</a>

Это подвал (footer) страниц сайта

Рис. 8.24. Фрагмент главной страницы сайта после удаления клиента с идентификатором `id=1`

Теперь нажмем на этой странице на ссылку Удалить для клиента Игорь Климов (`id = 1`) — сведения о нем исчезнут из таблицы (рис. 8.24).

Можно убедиться в том, что клиент с `id=1` действительно удален из БД, если открыть таблицу `firstapp_person` с помощью SQLiteStudio (рис. 8.25).

Итак, мы разобрались с тем, как с помощью моделей можно вносить данные в БД, редактировать и удалять информацию из БД через HTML-шаблоны Django, не прибегая к прямому использованию языка запросов SQL. Теперь рассмотрим возможности применения моделей и шаблонов для работы с изображениями и файлами.

	<b>id</b>	<b>name</b>	<b>age</b>
1	2	Житомирский Владимир	59
2	3	Шестов Алексей	55
3	4	Чурилко Николай	62

Рис. 8.25. Клиента с `id=1` после его удаления через HTML-страницу в БД больше нет

## 8.8. Работа с изображениями и файлами в формах Django

Загрузка в БД сайтов изображений и других файлов, например текстов в формате PDF, и их вывод на HTML-страницы — это неотъемлемые функции любого современного веб-приложения. К счастью, в Django реализованы простые процедуры работы с изображениями и другими файлами. Используя Django, можно загружать на сайт файлы различных типов и хранить их как в определенных каталогах сайта, так и непосредственно в таблицах БД. В этом разделе будет показано, как загружать на сайты файлы различных типов и выводить их на HTML-страницы.

### 8.8.1. Загрузка изображений

Рассмотрим процедуру загрузки на сайт изображений на простых примерах. Для этого создадим новую модель для описания структуры БД, форму для загрузки изображений, и шаблон HTML-страницы для загрузки изображений. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения изображений (листинг 8.14).

**Листинг 8.14. Измененный код модуля `models.py`**

```
# Загрузка изображений
class Image(models.Model):
    title = models.CharField(max_length=100, null=False,
                            verbose_name="Описание изображения",)
    image = models.ImageField(upload_to='images',
                            verbose_name="Файл с изображением",
                            null=True, blank=True)
    obj_img = models.Manager()
    def __str__(self):
        return self.title
```

Здесь была создана модель `Image`, в которой описаны два поля:

- `title` — заголовок, поясняющий содержание изображения;
- `image` — поле для хранения имени файла с изображением;
- `obj_img` — менеджер для работы с объектами изображений в модели.

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `image_file` может оставаться пустыми (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться изображения. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будет выполнена загрузка файлов. В данном случае файлы с изображениями будут храниться в папках `media/images/`.

К этому классу добавлен метод `def __str__`. Он позволит отобразить поле с описанием изображения в административной панели Django.

Для работы с изображениями необходимо установить библиотеку `pillow`, подав команду `pip`

```
pip install Pillow
```

Теперь добавим настройки в файл `hello/hello/settings.py` нашего проекта (листинг 8.15).

**Листинг 8.15. Измененный код модуля `settings.py`**

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

Здесь созданные переменные имеют следующий смысл:

- `MEDIA_URL` — это ссылочный URL-адрес для браузера, который обеспечивает доступ к файлам изображений через HTTP (базовый URL для обслуживания медиафайлов);
- `MEDIA_ROOT` — путь к каталогу, в котором будут храниться медиафайлы.

Изменения, внесенные в файл `settings.py`, показаны на рис. 8.26.

На следующем шаге необходимо изменить конфигурацию в файле `hello/hello/urls.py` (листинг 8.16, изменения выделены серым фоном).

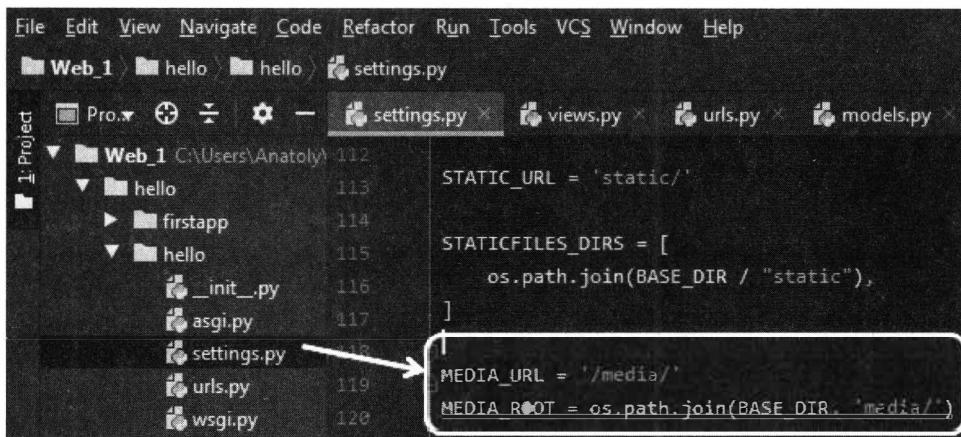


Рис. 8.26. Задание базового URL и пути к папке с медиафайлами в модуле settings.py

#### Листинг 8.16. Измененный код модуля urls.py

```

from django.contrib import admin
from django.urls import include, path
# добавлено для работы с медиа файлами
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('', include('firstapp.urls')),
    path('admin/', admin.site.urls),
]
# добавлено для работы с медиафайлами локально
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)

```

В этом программном коде добавлен импорт пакетов Django `settings` и `static`. Кроме того, к диспетчеру адресов `urlpatterns` добавлены ссылки на папки, в которых будут храниться загруженные пользователями медиафайлы. Здесь задана инструкция `if settings.DEBUG`, т. е. подключение этих адресов будет происходить только тогда, когда выполняется разработка и отладка программного кода. Это означает, что указанные URL-адреса будут использоваться только на этапе создания сайта на компьютере разработчика. При развертывании сайта на серверах в сети Интернет эти параметры можно изменить.

На данном этапе все необходимые настройки выполнены, и нужно запустить следующие команды для миграции созданной модели в БД:

```

python manage.py makemigrations;
python manage.py migrate

```

Если теперь открыть БД, то можно увидеть, что там присутствует таблица `firstapp_image` (рис. 8.27).

Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность
1 id	integer	key		
2 title	varchar (100)			
3 image	varchar (100)			

Рис. 8.27. Таблица БД firstapp\_image для хранения сведений о загруженных изображениях

Как видно из данного рисунка, в БД создана таблица firstapp\_image с тремя полями:

- id — идентификатор записи (тип поля integer — целое число);
- title — заголовок, описание изображения (тип поля varchar — текстовое поле переменной длины с максимальным числом символов 100);
- image — поле для хранения имени файла с изображением (тип поля varchar — текстовое поле переменной длины с максимальным числом символов 100).

Поскольку image — это текстовое поле, то в нем будет храниться только имя файла с изображением, но не сам файл.

Теперь создадим форму, которая будет принимать изображение в качестве входных данных от пользователя. Откроем файл hello/firstapp/forms.py и напишем в нем код листинга 8.17.

#### Листинг 8.17. Измененный код модуля forms.py

```
from .models import Image # для работы с изображениями
class ImageForm(forms.ModelForm):
    class Meta:
        model = Image
        fields = '__all__'
        # fields = ['title', 'image']
```

Здесь из модуля с моделями был импортирован класс Image, и создана наша форма — класс ImageForm на основе базового класса forms.ModelForm. Затем инициирован класс Meta, в котором созданы объекты model и fields. Преимущество создания формы на основе базового класса Django forms.ModelForm заключается в том, что форма автоматически создаст поля на HTML-странице в соответствии с полями модели Image, упомянутыми в модуле model.py. В инструкции fields = '\_\_all\_\_' указано, что на форме нужно будет отобразить все поля, которые присутствуют в модели. Поля можно было задать и в явном виде, как это делается показано в закомментированной строке.

Чтобы отобразить форму для загрузки изображений, создадим в папке с шаблонами Dgango HTML-страницу с именем hello/templates/firstapp/form\_up\_img.html (листинг 8.18).

**Листинг 8.18. Шаблон формы form\_up\_img.html**

```
{% extends "firstapp/base.html" %}  
{% block title %}Формы{% endblock title %}  
{% block header %}Изучаем формы Django{% endblock header %}  
{% block content %}  
<div class="container-fluid text-start my-2  
border border-5 border-warning">  
    <h5>Формы – загрузка изображений</h5>  
    <form method="POST" enctype="multipart/form-data">  
        {% csrf_token %}  
        <div class="form-group my-2">  
            {{ form.as_p }}  
            <button type="submit">Загрузить</button>  
        </div>  
    </form>  
    {% if my_img.count > 0 %}  
        <h3> {{my_text}}</h3>  
        <table class="table table-striped table-bordered  
                text-start">  
            <thead>  
                <tr>  
                    <th>№</th>  
                    <th>Что изображено</th>  
                    <th>Имя файла</th>  
                    <th>Изображение</th>  
                    <th>Удаление</th>  
                    <th>Показать</th>  
                </tr>  
            </thead>  
            <tbody>  
                {% for img in my_img %}  
                    <tr>  
                        <td>{{ img.id }}</td>  
                        <td>{{ img.title }}</td>  
                        <td>{{ img.image }}</td>  
                        <td></td>  
                        <td><a href="delete_img/{{img.id}}">Удалить</a></td>  
                        <td><a href="{{ img.image.url }}"  
                            class="btn btn-primary"  
                            target="_blank"> Показать</a></td>  
                    </tr>  
                {% endfor %}  
            </tbody>  
        </table>  
    {% endif %}  
</div>  
{% endblock content %}
```

Здесь у шаблона есть два важных тега:

- {{ form.as\_p }} — для отображения полей, обеспечивающих загрузку изображений;
- {% if my\_img.count > 0 %} — для показа изображений, которые пользователь загрузил в БД.

Здесь тег csrf\_token защитит форму от вредоносных данных, а form.as\_p отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### ПРИМЕЧАНИЕ

Здесь используется опция enctype = "multipart/form-data", которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Этую опцию необходимо обязательно включать в формы для работы с файлами и изображениями.

В теге {% if my\_img.count > 0 %} делается проверка, есть ли загруженные изображения. Если это условие выполняется, то в теге <table> будет создана таблица, в которой в цикле (тег {% for img in my\_img %}) будут показаны все изображения, которые были загружены на сайт и находятся по соответствующим ссылкам. Для этих изображений задан стиль (style="max-height:300px") — ограничение по высоте не более 300 px. В столбце таблицы Удаление сделана ссылка на функцию удаления изображения из БД. В столбце таблицы Показать находится кнопка со ссылкой на URL-адрес изображения, по которой оно будет показано в полном размере в отдельной вкладке браузера. Следовательно, в данной форме можно загружать, просматривать и удалять изображения.

Теперь создадим представление (view) для обработки запросов, связанных с загрузкой и удалением изображений. Откроем модуль hello/firstapp/views.py и напишем в нем код из листинга 8.19.

#### Листинг 8.19. Измененный код модуля views.py

```
# импорт модели и формы для работы с изображениями
from .models import Image
from .forms import ImageForm
# загрузка изображений
def form_up_img(request):
    if request.method == 'POST':
        form = ImageForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
    my_text = 'Загруженные изображения'
    my_img = Image.obj_img.all()
    form = ImageForm()
    context = {'my_text': my_text, "my_img": my_img, "form": form}
    return render(request, 'firstapp/form_up_img.html', context)
# удаление изображения из БД
def delete_img(request, id):
    try:
        img = Image.obj_img.get(id=id)
        img.delete()
    return redirect('form_up_img')
```

```
except Person.DoesNotExist:  
    return HttpResponseRedirect("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с изображениями:

```
from .models import Image  
from .forms import ImageForm
```

Затем создана функция для загрузки изображений — `def form_up_img(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку изображения (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `ImageForm` создается объект `form`, который получает запрос от пользователя на сохранение данных об изображении (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то выполняется сохранение введенных пользователем данных (`form.save()`). После этого происходит обновление формы для загрузки изображений.

Если форма вызывается первый раз, т. е. поступил запрос GET, то создаются:

- текстовая переменная `my_text`;
- объект `my_img`, который принимает из БД все сведения о загруженных изображениях;
- объект `form`, который создается на основе класса `ImageForm` (т. е. сама форма).

Затем создается объект `context`, в который в виде словаря передаются объекты: `my_text`, `my_img`, `form`. После этого вызывается шаблон `form_up_img.html`, в который передаются все данные через объект `context`.

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с ее URL-адресом в `urls.py`. Открываем модуль `hello/firstapp/urls.py` и добавляем в него две строки (листинг 8.20, изменения выделены серым фоном).

#### Листинг 8.20. Измененный код модуля `urls.py`

```
from django.urls import path  
from . import views  
urlpatterns = [  
    path('', views.index, name='index'),  
    path('about/', views.about, name='about'),  
    path('contact/', views.contact, name='contact'),  
    path('my_form/', views.my_form, name='my_form'),  
    path('my_form/edit_form/<int:id>', views.edit_form, name='edit_form'),  
    path('my_form/delete/<int:id>', views.delete),  
    path('form_up_img/', views.form_up_img, name='form_up_img'),  
    path('form_up_img/delete_img/<int:id>', views.delete_img),  
]
```

В этом программном коде была сопоставлена функция загрузки изображения (`views.form_up_img`) с URL-адресом шаблона HTML-страницы, которая отображает саму форму (`'form_up_img/'`), и функция удаления изображения (`views.delete_img`) с ее URL-адресом (`form_up_img/delete_img/<int:id>/`). В этой инструкции в функцию `delete_img` передается идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг — необходимо включить вызов формы `form_up_img.html` из меню главной страницы сайта. Для этого в базовый шаблон сайта `hello/templates/firstapp/base.html` добавим одну строку — ссылку на страницу `form_up_img.html`:

```
<a href="{% url 'form_up_img' %}">Изображения</a>
```

Полный код файла `base.html` приведен в листинге 8.21 (строка с изменениями выделена серым фоном).

### Листинг 8.21. Измененный код модуля `base.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
    <title>{% block title %}{% endblock title %}</title>
</head>
<body>
    <div class="container-fluid p-1 bg-primary text-white text-center">
        <div class="row">
            <div class="col-2 ">
                
            </div>
            <div class="col-10 ">
                <h1>Это заголовок главной страницы сайта</h1>
            </div>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row bg-warning text-center">
            <h6>
                <a href="{% url 'index' %}">Главная страница</a>
                <a href="{% url 'my_form' %}">Форма</a>
                <a href="{% url 'form_up_img' %}">Изображения</a>
                <a href="{% url 'about' %}">О компании</a>
                <a href="{% url 'contact' %}">Контакты</a>
            </h6>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row text-center text-primary fs-1 fw-bold">
            <div>{% block header%}{% endblock header %}</div>
        </div>
        <div class="row text-center text-body">
            <div>{% block content%}{% endblock content %}</div>
        </div>
    </div>
</body>
```

```
<div class="container-fluid">
    <div class="row bg-primary text-center text-white lh-1">
        <p>Это подвал (footer) страниц сайта</p>
    </div>
</div>
</body>
</html>
```

На данном этапе были запрограммированы все необходимые процедуры по загрузке и удалению изображений. Однако у Django есть одна особенность — когда запись о загруженном изображении удаляется из БД, то сам файл остается в хранилище. Для того чтобы не писать дополнительный программный код для физического удаления файлов, в Django имеется специальный пакет, но его нужно загрузить и подключить. Для загрузки пакета очистки файловых хранилищ выполним команду

```
pip install django-cleanup
```

После этого пакет django-cleanup нужно подключить к проекту. Открываем файл `hello/hello/settings.py` и добавляем в него одну строку (листинг 8.22, изменения выделены серым фоном).

#### Листинг 8.22. Измененный код модуля `settings.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'firstapp',
    'django_cleanup',
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку изображения в БД нашего учебного сайта. Запускаем наш веб-сервер, выполнив команду

```
python manage.py runserver
```



Рис. 8.28. Опция в меню сайта для вызова формы загрузки изображений

На главной странице сайта в главном меню появилась новая опция **Изображения** (рис. 8.28).

Если теперь щелкнуть мышью на ссылке **Изображения**, то откроется форма для загрузки изображений (рис. 8.29).

Это заголовок главной страницы сайта

Главная страница Форма Изображения О компании Контакты

## Изучаем формы Django

**Формы - загрузка изображений**

Описание изображения:

Файл с изображением:  Выберите файл | Файл не выбран

Это подвал (footer) страниц сайта

Рис. 8.29. Форма для загрузки изображений

Это заголовок главной страницы сайта

Главная страница Форма Изображения Файлы О компании Контакты

## Изучаем формы Django

**Формы - загрузка изображений**

Описание изображения:

Файл с изображением:  Выберите файл | Файл не выбран

**Загруженные изображения**

№	Что изображено	Имя файла	Изображение	Удаление	Показать
1	Академия Python	images/AP_400.png		<input type="button" value="Удалить"/>	<input type="button" value="Показать"/>

Это подвал (footer) страниц сайта

Рис. 8.30. Форма form\_up\_img.html после загрузки изображения

Как видно из рис. 8.29, на форме имеется поле для ввода текста с описанием изображения и две кнопки: **Выберите файл** — для выбора файла с изображением и **Загрузить** — для загрузки изображения в БД сайта.

Если теперь выбрать какой-нибудь файл с изображением, ввести его описание и нажать на кнопку **Загрузить**, то данный файл будет загружен на сайт, а имя загруженного файла будет сохранено в БД. После этих действий страница будет выглядеть так, как показано на рис. 8.30.

После этих действий в структуре проекта появятся две папки `media/images/`, где и будут сохраняться загружаемые изображения, а имена загруженных файлов запишутся в БД в таблицу `firstapp/image` (рис. 8.31).

Если нажать на кнопку **Показать**, то в браузере откроется новая вкладка, в которой будет показана не «миниатюра», а полноразмерное изображение (рис. 8.32).

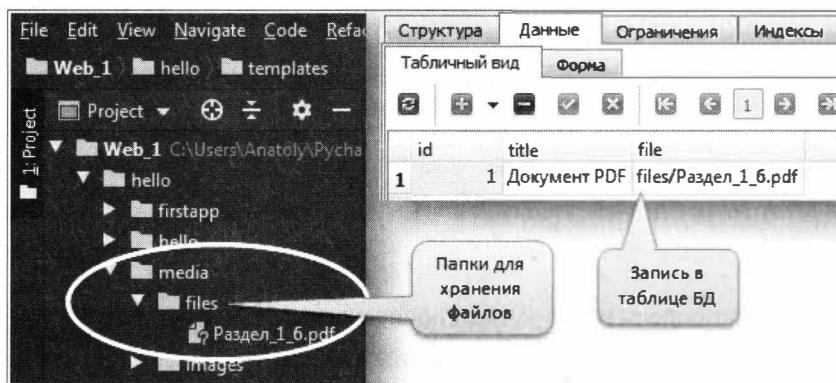


Рис. 8.31. Папки для хранения файлов и записи в таблице БД с именами загруженных файлов

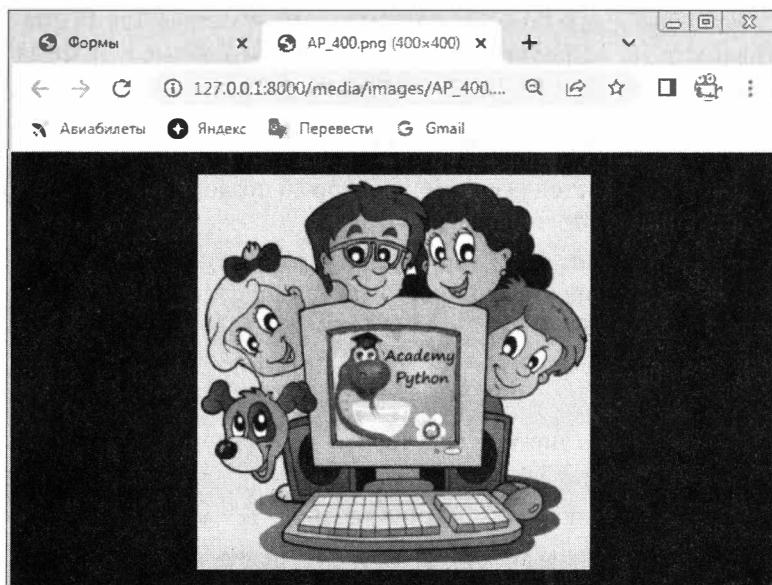


Рис. 8.32. Полноразмерное изображение в отдельной вкладке браузера

Таким образом, на сайте мы реализовали загрузку, просмотр и удаление изображений. Теперь рассмотрим выполнение аналогичных действий с файлами других типов.

## 8.8.2. Загрузка и отображение файлов PDF в формах Django

Рассмотрим процедуру загрузки на сайт файлов на примере документов в формате PDF. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки документов в формате PDF. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения файлов (листинг 8.23).

### Листинг 8.23. Измененный код модуля `models.py`

```
# Загрузка файлов
class File(models.Model):
    title = models.CharField(max_length=100,
                           verbose_name="Описание файла",)
    file = models.FileField(upload_to='files',
                           verbose_name="Файл PDF",
                           null=True, blank=True)

    def __str__(self):
        return self.title
```

Здесь была создана модель `File`, в которой описаны два поля:

- `title` — заголовок, поясняющий содержание файла;
- `file` — поле для хранения имени файла.

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `file` может оставаться пустыми (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будут загружены файлы. В данном случае файлы будут храниться в папках `media/files/`.

К этому классу добавлен метод `def __str__`, который позволит отобразить поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```
python manage.py makemigrations;
python manage.py migrate
```

Если теперь открыть БД, то можно увидеть, что там присутствует таблица `firstapp_file` (рис. 8.33).

Как видно из данного рисунка, в БД создана таблица `firstapp_file` с тремя полями:

- `id` — идентификатор записи (тип поля `integer` — целое число);
- `title` — заголовок, поясняющий содержание файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов, равным 100);

- ❑ `file` — поле для хранения имени файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов, равным 100).

Поскольку `file` — это текстовое поле, то в нем будет храниться только имя файла, но не сам файл.

Структура						Данные	Ограничения	Индексы	Триггеры
		Имя таблицы							
	Имя	Тип данных	Первичный ключ	Внешний ключ	Уникальность				
1	<b>id</b>	integer							
2	<b>title</b>	varchar (100)							
3	<b>file</b>	varchar (100)							

Рис. 8.33. Таблица БД `firstapp_file` для хранения сведений о загруженных файлах

Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл `hello/firstapp/forms.py` и напишем в нем код из листинга 8.24.

#### Листинг 8.24. Измененный код модуля `forms.py`

```
from .models import File # для работы с файлами
class FileForm(forms.ModelForm):
    class Meta:
        model = File
        fields = '__all__'
```

Здесь из моделей был импортирован класс `File`, затем в классе `Meta` на его основе созданы объекты `model` и `fields` (поля). В объект `fields` будут загружены все поля из модели `File`.

Теперь создадим представление (`view`) для обработки запросов, связанных с загрузкой файлов. Откроем модуль `hello/firstapp/views.py` и напишем в нем код листинга 8.25.

#### Листинг 8.25. Измененный код модуля `views.py`

```
# Импорт модели и формы для работы с файлами
from .models import File
from .forms import FileForm

# загрузка файлов pdf
def form_up_pdf(request):
    if request.method == 'POST':
        form = FileForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            my_text = 'Загруженные файлы'
            form = FileForm()
```

```

file_obj = File.objects.all()
context = {'my_text': my_text, "file_obj": file_obj, "form": form}
return render(request, 'firstapp/form_up_pdf.html', context)
# удаление файлов из БД
def delete_pdf(request, id):
    try:
        pdf = File.objects.get(id=id)
        pdf.delete()
        return redirect('form_up_pdf')
    except Person.DoesNotExist:
        return HttpResponseNotFound("<h2>Объект не найден</h2>")

```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с файлами:

```

from .models import File
from .forms import FileForm

```

Затем создана функция для загрузки файлов — `def form_up_pdf(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку файла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `FileForm` создается объект `form`, который получает запрос от пользователя на сохранение данных о файле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то введенные пользователем данные сохраняются (`form.save`). После этого происходит обновление формы для загрузки файлов.

Если форма вызывается первый раз, т. е. поступил запрос GET, то создаются:

- текстовая переменная `my_text`;
- объект `form`, который создается на основе класса `FileForm` (т. е. сама форма);
- объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создается объект `context`, в который в виде словаря передаются объекты `my_text`, `file_obj` и `form`. После этого вызывается шаблон `form_up_pdf.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки файлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_pdf.html` (листинг 8.26).

#### Листинг 8.26. Код шаблона `form_up_pdf.html`

```

{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}

```

```
<button type="submit">Загрузить</button>
</div>
</form>
{%
    if file_obj.count > 0 %}
    <h3> {{my_text}}</h3>
    <table class="table table-striped table-bordered
                text-start">
        <thead>
            <tr>
                <th> № </th>
                <th> Описание файла </th>
                <th> Имя файла </th>
                <th> Удаление </th>
                <th> Показать </th>
            </tr>
        </thead>
        <tbody>
            {% for obj in file_obj %}
            <tr>
                <td> {{ obj.id }} </td>
                <td> {{ obj.title }} </td>
                <td> {{ obj.file }} </td>
                <td> <a href="delete_pdf/{{obj.id}}">Удалить</a> </td>
                <td> <a href="{{ obj.file.url }}" class="btn btn-primary"
                    target="_blank"> Показать </a> </td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{%
    endif %}
</div>
{%
    endblock content %}
```

Здесь шаблон имеет два важных тега:

- {{ form.as\_p }} — для отображения полей, обеспечивающих загрузку файлов;
  - {% if file\_obj.count > 0 %} — для показа файлов, которые пользователь загрузил в БД.
- Здесь тег csrf\_token защитит форму от вредоносных данных, а form.as\_p отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### ПРИМЕЧАНИЕ

Здесь используется опция enctype = "multipart/form-data", которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Этую опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге {% if file\_obj.count > 0 %} делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге <tbl> будет создана таблица, в которой в цикле (тег {% obj in file\_obj %}) будут показаны все файлы, которые были загружены на сайт и находятся по соответствующим ссылкам. В колонке таблицы Удаление сделана ссылка на функцию удаления файла из БД. В колонке таблицы Показать находится кнопка со ссылкой на URL-адрес документа, по которому он будет показан в полном размере

в отдельной вкладке браузера. Иначе говоря, в данной форме можно загружать файлы PDF, просматривать их и удалять с сайта.

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с ее URL-адресом в urls.py. Открываем модуль hello/firstapp/urls.py и добавляем в него две строки (листиг 8.27, изменения выделены серым фоном).

#### Листинг 8.27. Измененный код модуля urls.py

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>/', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>/', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_img/delete_img/<int:id>/', views.delete_img),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
    path('form_up_pdf/delete_pdf/<int:id>/', views.delete_pdf),
]
```

В этом программном коде была сопоставлена функция загрузки файлов (views.form\_up\_pdf) с URL-адресом шаблона HTML-страницы, которая отображает саму форму ('form\_up\_pdf/'), и функция удаления файла (views.delete\_pdf) с ее URL-адресом (form\_up\_pdf/delete\_pdf/<int:id>/). В этой инструкции в функцию delete\_pdf передается идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг — необходимо включить вызов формы form\_up\_pdf.html из меню главной страницы сайта. Для этого в базовый шаблон сайта hello/templates/firstapp/base.html добавим одну строку — ссылку на страницу form\_up\_pdf.html. В листинге 8.28 приведен фрагмент кода модуля base.html, где добавленная строка выделена серым фоном.

#### Листинг 8.28. Измененный код модуля base.html

```
<div class="row bg-warning text-center">
<h6>
    <a href="{% url 'index' %}">Главная страница</a>
    <a href="{% url 'my_form' %}">Форма</a>
    <a href="{% url 'form_up_img' %}">Изображения</a>
    <a href="{% url 'form_up_pdf' %}">Файлы</a>
    <a href="{% url 'about' %}">О компании</a>
    <a href="{% url 'contact' %}">Контакты</a>
</h6>
</div>
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку файлов PDF в БД нашего учебного сайта. Запускаем наш веб-сервер командой

```
python manage.py runserver
```

На главной странице сайта в главном меню появилась новая опция **Файлы** (рис. 8.34). Если теперь щелкнуть мышью на ссылке **Файлы**, то откроется форма для загрузки файлов (рис. 8.35).

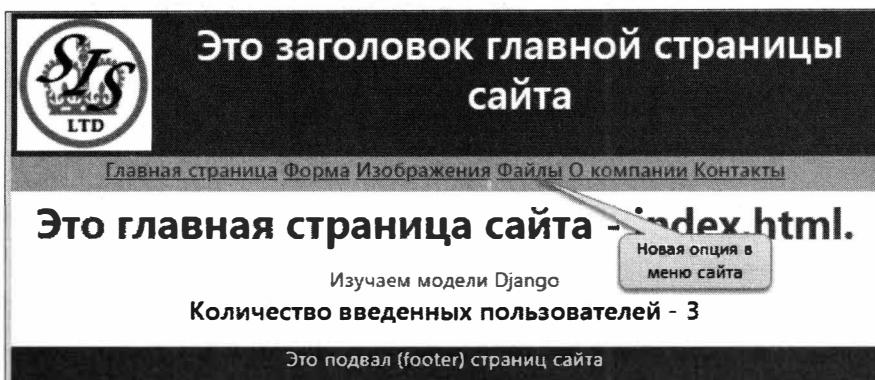


Рис. 8.34. Опция в меню сайта вызова формы для загрузки файлов

A screenshot of a file upload form. The title 'Это заголовок главной страницы сайта' is at the top. Below it, the title 'Изучаем формы Django' is displayed. A section titled 'Формы - загрузка файлов' contains a text input field labeled 'Описание файла:' and a file selection input field labeled 'Файл PDF: Выберите файл'. Below these fields is a button labeled 'Загрузить'. At the bottom of the form, a footer section contains the text 'Это подвал (footer) страниц сайта'.

Рис. 8.35. Форма для загрузки файлов

Как видно из данного рисунка, на форме имеется поле для ввода текста с описанием файла и две кнопки: **Выберите файл** — для выбора файла и **Загрузить** — для загрузки файла в БД сайта.

Если теперь выбрать какой-нибудь файл (документ в формате PDF), ввести его описание и нажать на кнопку **Загрузить**, то данный файл будет загружен на сайт, а имя загруженного сайта будет сохранено в БД. После этих действий страница будет выглядеть так, как показано на рис. 8.36.

После этих действий в структуре проекта появится папка для хранения файлов `media/files/`, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу `firstapp/file` (рис. 8.37).

Это заголовок главной страницы сайта

Главная страница Форма Изображения Файлы О компании Контакты

## Изучаем формы Django

**Формы - загрузка файлов**

Описание файла:

Файл PDF:  Выберите файл! Файл не выбран

**Загруженные файлы**

№	Описание файла	Имя файла	Удаление	Показать
1	Документ PDF	files/Раздел_1_6.pdf	<a href="#">Удалить</a>	<a href="#">Показать</a>

Это подвал (footer) страниц сайта

Рис. 8.36. Форма form\_up\_pdf.html после загрузки файла

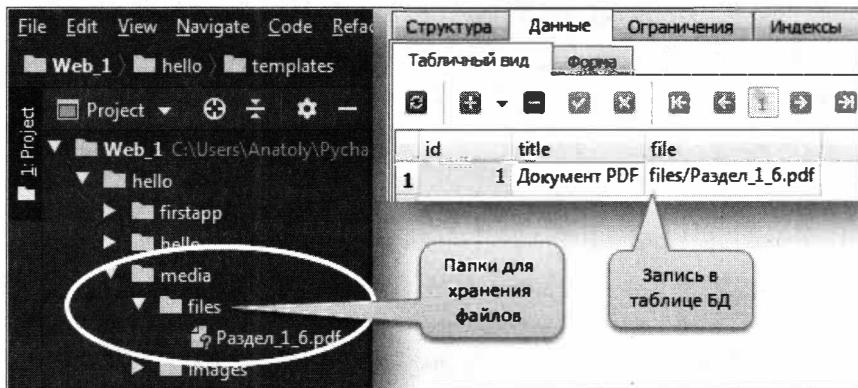


Рис. 8.37. Папки для хранения файлов PDF записи в таблице БД с именами загруженных файлов

Если нажать на кнопку **Показать**, то в браузере откроется новая вкладка, в которой будет показано содержание документа в формате PDF (рис. 8.38).

Как видно из рис. 8.38, на данной вкладке можно выполнять постраничное пролистывание документа, масштабирование, поворот страниц и вывод на печать. Таким образом, мы реализовали загрузку на сайт файлов формата PDF, просмотр документов и их удаление с сайта. Теперь рассмотрим выполнение аналогичных действий с файлами других типов.

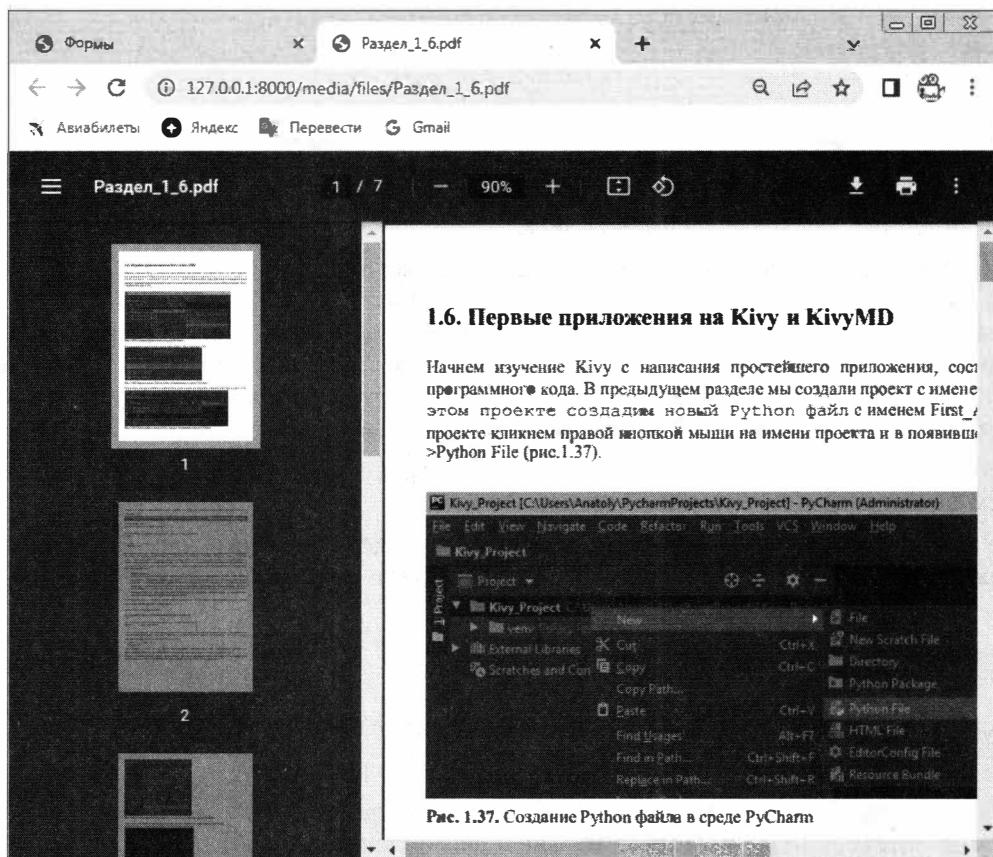


Рис. 8.38. Вывод содержимого PDF-файла в отдельной вкладке браузера

### 8.8.3. Загрузка и отображение видеофайлов в формах Django

Рассмотрим процедуру загрузки на сайт видеофайлов на примере коротких роликов в формате MP4. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки видеофайлов. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения видеофайлов (листинг 8.29).

#### Листинг 8.29. Измененный код модуля `models.py`

```
# Загрузка видеофайлов
class VideoFile(models.Model):
    title = models.CharField(max_length=100,
                           verbose_name="Описание файла",)
    file = models.FileField(upload_to='videos',
                           verbose_name="Видеофайл",
                           null=True, blank=True)
```

```
obj_video = models.Manager()
def __str__(self):
    return self.title
```

Здесь была создана модель `VideoFile`, в которой описаны два поля:

- `title` — заголовок, поясняющий содержание файла;
- `file` — поле для хранения имени файла.

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `file` может оставаться пустым (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будет выполнена загрузка файлов. В данном случае файлы будут храниться в папках `media/videos/`.

К этому классу добавлен метод `def __str__`, который позволит отобразить поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```
python manage.py makemigrations;
python manage.py migrate
```

После этого в БД будет создана таблица `firstapp_videofile` с тремя полями:

- `id` — идентификатор записи (тип поля `integer` — целое число);
- `title` — заголовок, поясняющий содержание файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100);
- `file` — поле для хранения имени файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100).

Поскольку `file` — это текстовое поле, то в нем будет храниться только имя файла, но не сам файл.

Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл `hello/firstapp/forms.py` и напишем в нем код листинга 8.30.

### Листинг 8.30. Измененный код модуля `forms.py`

```
from .models import VideoFile # для работы с файлами видео
class VideoForm(forms.ModelForm):
    class Meta:
        model = VideoFile
        fields = '__all__'
```

Здесь из моделей был импортирован класс `VideoFile`, затем в классе `Meta` на его основе созданы объекты `model` и `fields` (поля). В объект `fields` будут загружены все поля из модели `VideoFile`.

Теперь создадим представление (`view`) для обработки запросов, связанных с загрузкой файлов. Откроем модуль `hello/firstapp/views.py` и напишем в нем код листинга 8.31.

**Листинг 8.31. Измененный код модуля views.py**

```
# импорт модели и формы для работы с видеофайлами
from .models import VideoFile
from .forms import VideoForm
# загрузка видеофайлов
def form_up_video(request):
    if request.method == 'POST':
        form = VideoForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
    my_text = 'Загруженные видеофайлы'
    form = VideoForm()
    file_obj = VideoFile.obj_video.all()
    context = {'my_text': my_text, "file_obj": file_obj, "form": form}
    return render(request, 'firstapp/form_up_video.html', context)
# удаление видеофайлов из БД
def delete_video(request, id):
    try:
        video = VideoFile.obj_video.get(id=id)
        video.delete()
        return redirect('form_up_video')
    except Person.DoesNotExist:
        return HttpResponseRedirect("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с видеофайлами:

```
from .models import VideoFile
from .forms import VideoForm
```

Затем создана функция для загрузки файлов — `def form_up_video(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку видеофайла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `VideoForm` создается объект `form`, который получает запрос от пользователя на сохранение данных о видеофайле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то выполняется сохранение введенных пользователем данных (`form.save()`). После этого происходит обновление формы для загрузки видеофайлов.

Если форма вызывается первый раз, т. е. поступил запрос GET, то создаются:

- текстовая переменная `my_text`;
- объект `form`, который создается на основе класса `VideoForm` (т. е. сама форма);
- объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создается объект `context`, в который в виде словаря передаются объекты: `my_text`, `file_obj`, `form`. После этого вызывается шаблон `form_up_video.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки видеофайлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_video.html` (листинг 8.32).

**Листинг 8.32. Шаблон form\_up\_video.html**

```
{% extends "firstapp/base.html" %}  
{% block title %}Формы{% endblock title %}  
{% block header %}Изучаем формы Django{% endblock header %}  
{% block content %}  
<div class="container-fluid text-start my-2  
    border border-5 border-warning">  
    <h5>Формы – загрузка видеофайлов</h5>  
    <form method="POST" enctype="multipart/form-data">  
        {% csrf_token %}  
        <div class="form-group my-2">  
            {{ form.as_p }}  
            <button type="submit">Загрузить</button>  
        </div>  
    </form>  
    {% if file_obj.count > 0 %}  
        <h3> {{my_text}}</h3>  
        <table class="table table-striped table-bordered  
               text-start">  
            <thead>  
                <tr>  
                    <th>№</th>  
                    <th>Описание файла</th>  
                    <th>Имя файла</th>  
                    <th>Удаление</th>  
                    <th>Показать</th>  
                </tr>  
            </thead>  
            <tbody>  
                {% for obj in file_obj %}  
                    <tr>  
                        <td>{{ obj.id }}</td>  
                        <td>{{ obj.title }}</td>  
                        <td>{{ obj.file }}</td>  
                        <td><a href="delete_video/{{obj.id}}">Удалить</a></td>  
                        <td>  
                            <video width="320" height="240" controls>  
                                <source src="{{obj.file.url}}"  
                                       type="video/mp4">  
                            </video>  
                        </td>  
                    </tr>  
                {% endfor %}  
            </tbody>  
        </table>  
    {% endif %}  
</div>  
{% endblock content %}
```

Здесь в шаблоне есть два важных тега:

- {{ form.as\_p }} — для отображения полей, обеспечивающих загрузку файлов;
- {% if file\_obj.count > 0 %} — для показа файлов, которые пользователь загрузил в БД.

Здесь тег csrf\_token защитит форму от вредоносных данных, а form.as\_p отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### ПРИМЕЧАНИЕ

Здесь используется опция enctype = "multipart/form-data", которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Эту опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге {% if file\_obj.count > 0 %} делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге <tbl> будет создана таблица, в которой в цикле (тег {% obj in file\_obj %}) будут показаны все файлы, загруженные на сайт и находящиеся по соответствующим ссылкам. В колонке таблицы Удаление сделана ссылка на функцию удаления файла из БД. В колонке таблицы Показать находится кнопка со ссылкой на URL-адрес видеофайла, по которому он будет показан в данной колонке. В результате в данной форме можно загружать видеофайлы, просматривать их и удалять с сайта.

Здесь следует обратить особое внимание на тег <video>. До появления HTML 5 веб-разработчикам приходилось вставлять видео на веб-страницу с помощью подключаемого модуля, такого как Adobe Flash Player. С помощью Bootstrap 5 можно легко вставлять видео в HTML-документ с помощью тега <video>.

Тег <video> поддерживает глобальные атрибуты, которые используются всеми тегами HTML, такие как id, class, style и т. д. Кроме того, он имеет набор собственных атрибутов: autoplay controls, loop, muted, src, poster, preload и др. Рассмотрим наиболее важные атрибуты.

**Атрибут src** используется для указания источника видео. Это может быть относительный путь к видео на локальном компьютере или ссылка на видео из Интернета:

```
<video src="weekend.mp4"></video>
```

Вместо него можно использовать тег <source>.

**Атрибут poster** позволяет добавить изображение, которое будет показано до начала воспроизведения видео или во время его загрузки.

```
<video src="weekend.mp4" poster="benefits-of-coding.jpg"></video>
```

Вместо изображения первой сцены из видео браузер покажет это изображение.

**Атрибут controls.** Когда используется элемент control, то в браузере будут отображаться контроллеры управления воспроизведением, такие как "пуск", "пауза", "громкость", "поиск" и т. д. Например:

```
<video  
    controls  
    src="weekend.mp4"  
    poster="benefits-of-coding.jpg"  
></video>
```

**Атрибут `loop`** дает возможность автоматически повторять видео — заставлять его воспроизводиться снова каждый раз, когда показ завершится. Например:

```
<video  
    controls  
    loop  
    src="weekend.mp4"  
    poster="benefits-of-coding.jpg"  
></video>
```

**Атрибут `autoplay`** позволяет настроить автоматическое воспроизведение видео сразу после загрузки страницы. Например:

```
<video  
    controls  
    loop  
    autoplay  
    src="weekend.mp4"  
    poster="benefits-of-coding.jpg"  
></video>
```

**Атрибуты `width` и `height`** дают возможность указать ширину и высоту видео в пикселях. Он принимает только абсолютные значения, например:

```
<video  
    controls  
    loop  
    autoplay  
    src="weekend.mp4"  
    width="350px"  
    height="250px"  
    poster="benefits-of-coding.jpg"  
></video>
```

**Атрибут `muted`** можно использовать для того, чтобы браузер не воспроизводил звуковую дорожку, связанную с видео. Например:

```
<video  
    controls  
    loop  
    autoplay  
    muted  
    src="weekend.mp4"  
    width="350px"  
    height="250px"  
    poster="benefits-of-coding.jpg"  
></video>
```

**Атрибут `preload`** "подскажет" браузеру, следует ли загружать видео при загрузке страницы. Этот атрибут имеет решающее значение для взаимодействия с пользователем. Доступны следующие значения этого атрибута:

- none** — указывает, что видео не будет загружаться, пока пользователь не нажмет кнопку воспроизведения;

- auto — указывает, что видео должно загружаться, даже если пользователь не нажимает кнопку воспроизведения;
- metadata — указывает, что браузер должен собирать метаданные, такие как длина, размер, продолжительность и т. д.

Вот пример использования этого атрибута:

```
<video
    controls
    loop
    autoplay
    muted="true"
    preload="metadata"
    src="weekend.mp4"
    width="350px"
    height="250px"
    poster="benefits-of-coding.jpg"
></video>
```

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с ее URL-адресом в urls.py. Открываем модуль hello/firstapp/urls.py и добавляем в него две строки (листинг 8.33, изменения выделены серым фоном).

#### Листинг 8.33. Измененный код модуля urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('my_form/', views.my_form, name='my_form'),
    path('my_form/edit_form/<int:id>', views.edit_form, name='edit_form'),
    path('my_form/delete/<int:id>', views.delete),
    path('form_up_img/', views.form_up_img, name='form_up_img'),
    path('form_up_img/delete_img/<int:id>', views.delete_img),
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),
    path('form_up_pdf/delete_pdf/<int:id>', views.delete_pdf),
    path('form_up_video/', views.form_up_video, name='form_up_video'),
    path('form_up_video/delete_video/<int:id>', views.delete_video),
]
```

В этом программном коде была сопоставлена функция загрузки видеофайлов (views.form\_up\_video) с URL-адресом шаблона HTML-страницы, которая отображает саму форму ('form\_up\_video/'), и функция удаления файла (views.delete\_video) с ее URL-адресом (form\_up\_pdf/delete\_video/<int:id>/). В этой инструкции в функцию delete\_video передается идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг — необходимо включить вызов формы form\_up\_video.html из меню главной страницы сайта. Для этого в базовый шаблон сайта hello/templates/firstapp/base.html добавим одну строку — ссылку на страницу form\_up\_video.html. В листинге 8.34 приведен фрагмент кода модуля base.html, где добавленная строка выделена серым фоном.

### Листинг 8.34. Измененный код модуля base.html

```
<a href="{% url 'index' %}">Главная страница</a>
<a href="{% url 'my_form' %}">Форма</a>
<a href="{% url 'form_up_img' %}">Изображения</a>
<a href="{% url 'form_up_pdf' %}">Файлы</a>
<a href="{% url 'form_up_video' %}">Видео</a>
<a href="{% url 'about' %}">О компании</a>
<a href="{% url 'contact' %}">Контакты</a>
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку видеофайлов в БД нашего учебного сайта. Запускаем наш веб-сервер, выполнив команду

```
python manage.py runserver
```

На главной странице сайта в главном меню появилась новая опция **Видео** (рис. 8.39).

Если теперь щелкнуть мышью на ссылке **Видео**, то откроется форма для загрузки видеофайлов (рис. 8.40).



Рис. 8.39. Опция в меню сайта вызова формы для загрузки видеофайлов

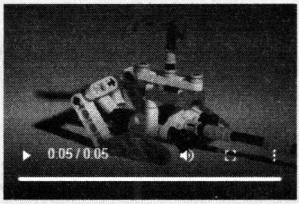
The screenshot shows a form titled 'Формы - загрузка видео файлов'. It includes a label 'Описание файла:' followed by an input field, a label 'ВидеоФайл:' followed by a file selection input field containing the placeholder 'Выберите файл', and a message 'Файл не выбран'. At the bottom of the form is a button labeled 'Загрузить'. The footer of the page is visible at the bottom of the screenshot.

Рис. 8.40. Форма для загрузки видеофайлов

Как видно из рис. 8.40, на форме имеется поле для ввода текста с описанием файла и две кнопки: **Выберите файл** — для выбора файла, **Загрузить** — для загрузки файла в БД сайта.

Если теперь выбрать какой-нибудь видеофайл (в формате MP4), ввести его описание и нажать на кнопку **Загрузить**, то данный файл будет загружен на сайт, а имя загруженного сайта будет сохранено в БД. После этих действий страница будет выглядеть так, как показано на рис. 8.41.

The screenshot shows a Django application interface. At the top, there's a navigation bar with links: Главная страница, Формы, Контакты, Файлы, Видео, О компании, Контакты. Below the navigation, the title "Это заголовок главной страницы сайта" is displayed. The main content area has a title "Изучаем формы Django". A form titled "Формы - загрузка видео файлов" contains a text input field labeled "Описание файла:" and a file input field labeled "Видео файл: Выберите файл" with the placeholder "Файл не выбран". Below the form is a button labeled "Загрузить". Underneath, a section titled "Загруженные видео файлы" lists one item:

Нº файла	Описание	Имя файла	Удаление	Показать
1	Конструктор	videos/small.mp4	<a href="#">Удалить</a>	 0.05 / 0.05

At the bottom of the page, the footer text "Это подвал (footer) страниц сайта" is visible.

Рис. 8.41. Форма form\_up\_video.html после загрузки файла

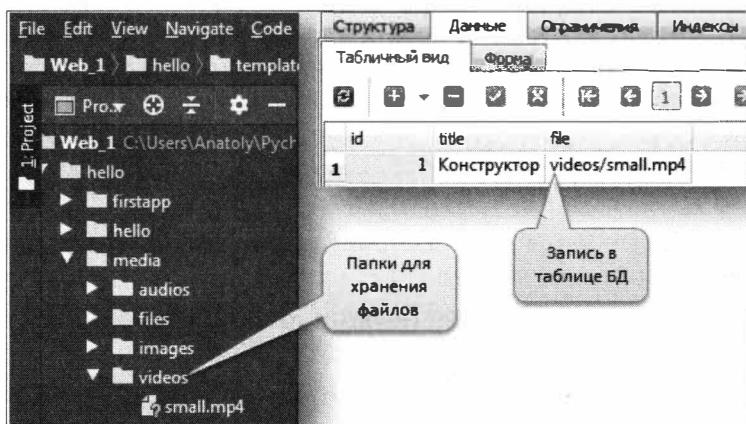


Рис. 8.42. Папки для хранения видеофайлов записи в таблице БД с именами загруженных файлов

После этих действий в структуре проекта будет создана папка для хранения видеофайлов `media/videos/`, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу `firstapp/videofile` (рис. 8.42).

Таким образом, мы реализовали загрузку на сайт видеофайлов, просмотр видео и их удаление с сайта. Теперь рассмотрим выполнение аналогичных действий с аудиофайлами.

### 8.8.4. Загрузка и озвучивание аудиофайлов в формах Django

Рассмотрим процедуру загрузки на сайт аудиофайлов на примере коротких роликов в формате MP3. Для этого создадим новую модель для описания структуры БД, форму для загрузки файлов и шаблон HTML-страницы для загрузки аудиофайлов. Откроем модуль `hello/firstapp/models.py` и создадим новую модель с описанием структуры таблицы БД для хранения аудиофайлов (листинг 8.35).

#### Листинг 8.35. Измененный код модуля `models.py`

```
# Загрузка аудиофайлов
class AudioFile(models.Model):
    title = models.CharField(max_length=100,
                           verbose_name="Описание файла",)
    file = models.FileField(upload_to='audios',
                           verbose_name="Аудиофайл",
                           null=True, blank=True)

    obj_audio = models.Manager()

    def __str__(self):
        return self.title
```

Здесь была создана модель `AudioFile`, в которой описаны два поля:

- `title` — заголовок, поясняющий содержание файла;
- `file` — поле для хранения имени файла;

Для обоих полей заданы метки (`verbose_name`), а также определено, что поле `title` обязательно для заполнения (`null=False`), а поле `file` может оставаться пустым (`null=True, blank=True`). Параметр `upload_to` указывает расположение файлов, т. е. каталог, в котором будут храниться файлы. Нам не нужно создавать каталог мультимедиа для хранения файлов вручную, он появится автоматически, когда будут загружены файлы. В данном случае файлы будут храниться в папках `media/audios/`.

К этому классу добавлен метод `def __str__`, который отобразит поле с описанием файлов в административной панели Django.

Все необходимые настройки были выполнены в разделе, описывающем загрузку изображений, и на данном этапе нужно просто запустить следующие команды для миграции созданной модели в БД:

```
python manage.py makemigrations;
python manage.py migrate
```

После этого в БД будет создана таблица `firstapp_audofile` с тремя полями:

- `id` — идентификатор записи (тип поля `integer` — целое число);
- `title` — заголовок, поясняющий содержание файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100);
- `file` — поле для хранения имени файла (тип поля `varchar` — текстовое поле переменной длины с максимальным числом символов 100).

Поскольку `file` — это текстовое поле, то в нем будет храниться только имя файла, но не сам файл.

Теперь создадим форму, которая будет принимать файл в качестве входных данных от пользователя. Откроем файл `hello/firstapp/forms.py` и напишем в нем код листинга 8.36.

#### Листинг 8.36. Измененный код модуля `forms.py`

```
from .models import AudioFile # для работы с файлами аудио
class AudioForm(forms.ModelForm):
    class Meta:
        model = AudioFile
        fields = '__all__'
```

Здесь из моделей был импортирован класс `AudioFile`, затем в классе `Meta` на его основе созданы объекты `model` и `fields` (поля). В объект `fields` будут загружены все поля из модели `AudioFile`.

Теперь создадим представление (`view`) для обработки запросов, связанных с загрузкой файлов. Откроем модуль `hello/firstapp/views.py` и напишем в нем код листинга 8.37.

#### Листинг 8.37. Измененный код модуля `views.py`

```
# импорт модели и формы для работы с аудиофайлами
from .models import AudioFile
from .forms import AudioForm

# Загрузка аудиофайлов
def form_up_audio(request):
    if request.method == 'POST':
        form = AudioForm(request.POST, request.FILES)
        if form.is_valid():
            form.save()
            my_text = 'Загруженные аудиофайлы'
            form = AudioForm()
            file_obj = AudioFile.obj_audio.all()
            context = {'my_text': my_text, "file_obj": file_obj, "form": form}
            return render(request, 'firstapp/form_up_audio.html', context)
    # удаление аудиофайлов из БД
def delete_audio(request, id):
    try:
        audio = AudioFile.obj_audio.get(id=id)
        audio.delete()
        return redirect('form_up_audio')
```

```
except Person.DoesNotExist:
    return HttpResponseRedirect("<h2>Объект не найден</h2>")
```

В этом модуле на первом шаге выполнен импорт модели и формы для работы с аудиофайлами:

```
from .models import AudioFile
from .forms import AudioForm
```

Затем создана функция для загрузки файлов — `def form_up_audio(request)`. В этой функции проверяется условие, поступил ли запрос от пользователя на загрузку аудиофайла (`if request.method == 'POST'`). Если такой запрос поступил, то на основе класса `AudioForm` создается объект `form`, который получает запрос от пользователя на сохранение данных об аудиофайле (`request.POST`), и сам загружаемый файл (`request.FILES`). Если форма не содержит ошибок, то выполняется сохранение введенных пользователем данных (`form.save()`). После этого происходит обновление формы для загрузки аудиофайлов.

Если форма вызывается первый раз, т. е. поступил запрос GET, то создаются:

- текстовая переменная `my_text`;
- объект `form`, который создается на основе класса `AudioForm` (т. е. сама форма);
- объект `file_obj`, который принимает из БД все сведения о загруженных файлах.

Затем создается объект `context`, в который в виде словаря передаются объекты `my_text`, `file_obj`, `form`. После этого вызывается шаблон `form_up_audio.html`, в который передаются все данные через объект `context`.

Чтобы отобразить форму для загрузки аудиофайлов, создадим в папке с шаблонами Django HTML-страницу с именем `hello/templates/firstapp/form_up_audio.html` (листинг 8.38).

### Листинг 8.38. Шаблон `form_up_audio.html`

```
{% extends "firstapp/base.html" %}
{% block title %}Формы{% endblock title %}
{% block header %}Изучаем формы Django{% endblock header %}
{% block content %}


Формы – загрузка аудиофайлов
<form method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <div class="form-group my-2">
        {{ form.as_p }}
        <button type="submit">Загрузить</button>
    </div>
</form>
{% if file_obj.count > 0 %}
    <h3> {{my_text}} </h3>
    <table class="table table-striped table-bordered
text-start">
        <thead>
            <tr>
                <th> № </th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td> {{file_obj}} </td>
            </tr>
        </tbody>
    </table>
</div>


```

```
<th>Описание файла</th>
<th>Имя файла</th>
<th>Удаление</th>
<th>Прослушать</th>
</tr>
</thead>
<tbody>
    {% for obj in file_obj %}
        <tr>
            <td>{{ obj.id }}</td>
            <td>{{ obj.title }}</td>
            <td>{{ obj.file }}</td>
            <td><a href="/delete_audio/{{obj.id}}">Удалить</a></td>
            <td>
                <audio controls>
                    <source src="{{obj.file.url}}"
                           type="audio/mp3">
                </audio>
            </td>
        </tr>
    {% endfor %}
</tbody>
</table>
{% endif %}
</div>
{% endblock content %}
```

Здесь в шаблоне есть два важных тега:

- {{ form.as\_p }} — для отображения полей, обеспечивающих загрузку файлов;
- {% if file\_obj.count > 0 %} — для показа файлов, которые пользователь загрузил в БД.

Здесь тег csrf\_token защитит форму от вредоносных данных, а form.as\_p отобразит поля, предназначенные для ввода данных, в виде параграфов.

#### **ПРИМЕЧАНИЕ.**

Здесь используется опция enctype = "multipart/form-data", которая позволяет отправлять файлы через POST-запросы. Без активации этой опции пользователь не может отправить файл через запрос POST. Эту опцию необходимо обязательно включать в формы работы с файлами и изображениями.

В теге {% if file\_obj.count > 0 %} делается проверка, есть ли загруженные файлы. Если это условие выполняется, то в теге <tbl> будет создана таблица, в которой в цикле (тег {% obj in file\_obj %}) будут показаны все файлы, которые были загружены на сайт и находятся по соответствующим ссылкам. В столбце таблицы Удаление сделана ссылка на функцию удаления файла из БД. В столбце таблицы Прослушать находится кнопка со ссылкой на URL-адрес аудиофайла, по которому он будет запущен на прослушивание. Таким образом, в данной форме можно загружать аудиофайлы, прослушивать их и удалять с сайта.

Здесь следует обратить особое внимание на тег <audio>, который используется для встраивания звукового контента на HTML-страницы. Он может содержать один или

более источников аудио, представленных с помощью атрибута `src` или элемента `<source>` — браузер выберет один наиболее подходящих.

**Атрибут `src`** указывает источник аудио. Это может быть относительный путь к аудио на локальном компьютере или ссылка на аудио из Интернета:

```
<audio src="weekend.mp3"></audio>
```

Его можно заменить тегом `<source>`.

**Атрибут `controls`.** Когда используется атрибут `control`, то в браузере будут отображаться контроллеры управления воспроизведением, такие как "пуск", "пауза", "громкость". Например:

```
<audio  
    controls  
    src="weekend.mp3"  
    poster="benefits-of-coding.jpg"  
></audio>
```

**Атрибут `loop`** дает возможность автоматически повторять аудио — заставлять его воспроизводиться снова каждый раз, когда оно завершится. Например:

```
<audio  
    controls  
    loop  
    src="weekend.mp3"  
></audio>
```

**Атрибут `autoplay`** позволяет настроить автоматическое воспроизведение аудио сразу после загрузки страницы. Например:

```
<audio  
    controls  
    loop  
    autoplay  
    src="weekend.mp3"  
></audio>
```

**Атрибут `muted`** можно использовать для того, чтобы браузер не воспроизводил звук. Например:

```
<audio  
    controls  
    loop  
    autoplay  
    muted  
    src="weekend.mp3"  
></audio>
```

**Атрибут `preload`** "подскажет" браузеру, следует ли загружать аудио при загрузке страницы. Этот атрибут имеет решающее значение для взаимодействия с пользователем. Возможны следующие значения атрибута предварительной загрузки:

- `none` — указывает, что аудио не будет загружаться, пока пользователь не нажмет кнопку воспроизведения;

- auto — указывает, что аудио должно загружаться, даже если пользователь не нажимает кнопку воспроизведения;
- metadata — указывает, что браузер должен собирать метаданные, такие как размер, продолжительность и т. д.

Вот пример использования этого атрибута:

```
<audio  
    controls  
    loop  
    autoplay  
    muted="true"  
    preload="metadata"  
    src="weekend.mp3"  
></audio >
```

Теперь, когда и форма, и обработчик формы готовы, сопоставим форму с ее URL-адресом в urls.py. Открываем модуль hello/firstapp/urls.py и добавляем в него две строки (листинг 8.39, изменения выделены серым фоном).

#### Листинг 8.39. Измененный код модуля urls.py

```
from django.urls import path  
from .import views  
urlpatterns = [  
    path('', views.index, name='index'),  
    path('about/', views.about, name='about'),  
    path('contact/', views.contact, name='contact'),  
    path('my_form/', views.my_form, name='my_form'),  
    path('my_form/edit_form/<int:id>/', views.edit_form, name='edit_form'),  
    path('my_form/delete/<int:id>/', views.delete),  
    path('form_up_img/', views.form_up_img, name='form_up_img'),  
    path('form_up_img/delete_img/<int:id>/', views.delete_img),  
    path('form_up_pdf/', views.form_up_pdf, name='form_up_pdf'),  
    path('form_up_pdf/delete_pdf/<int:id>/', views.delete_pdf),  
    path('form_up_video/', views.form_up_video, name='form_up_video'),  
    path('form_up_video/delete_video/<int:id>/', views.delete_video),  
    path('form_up_audio/', views.form_up_audio, name='form_up_audio'),  
    path('form_up_audio/delete_audio/<int:id>/', views.delete_audio),  
]
```

В этом программном коде была сопоставлена функция загрузки аудиофайлов (views.form\_up\_audio) с URL-адресом шаблона HTML-страницы, которая отображает саму форму ('form\_up\_audio/'), и функция удаления файла (views.delete\_audio) с ее URL-адресом (form\_up\_pdf/delete\_audio/<int:id>/). В этой инструкции в функцию delete\_audio передается идентификатор той записи, которую нужно удалить из БД.

Остался последний шаг — необходимо включить вызов формы form\_up\_audio.html из меню главной страницы сайта. Для этого в базовый шаблон сайта hello/templates/firstapp/base.html добавим одну строку — ссылку на страницу form\_up\_audio.html. В листинге 8.40 приведен фрагмент кода модуля base.html, где добавленная строка выделена серым фоном.

#### Листинг 8.40. Измененный код модуля base.html

```
<a href="{% url 'index' %}">Главная страница</a>
<a href="{% url 'my_form' %}">Форма</a>
<a href="{% url 'form_up_img' %}">Изображения</a>
<a href="{% url 'form_up_pdf' %}">Файлы</a>
<a href="{% url 'form_up_video' %}">Видео</a>
<a href="{% url 'form_up_audio' %}">Аудио</a>
<a href="{% url 'about' %}">О компании</a>
<a href="{% url 'contact' %}">Контакты</a>
```

Итак, у нас все готово для того, чтобы продемонстрировать загрузку аудиофайлов в БД нашего учебного сайта. Запускаем наш веб-сервер, выполнив команду

```
python manage.py runserver
```

На главной странице сайта в главном меню появилась новая опция **Аудио** (рис. 8.43).

Если теперь щелкнуть мышью на ссылке **Аудио**, то откроется форма для загрузки аудиофайлов (рис. 8.44).



Рис. 8.43. Опция в меню сайта вызова формы для загрузки аудиофайлов

The screenshot shows a form for uploading audio files. The title 'Это заголовок главной страницы сайта' is at the top. Below it, the text 'Изучаем формы Django' is displayed. The form itself has a title 'Формы - загрузка аудио файлов'. It contains a text input field labeled 'Описание файла:' with an empty input box. Below that is another input field labeled 'Аудио файл:' containing the text 'Выберите файл' and 'Файл не выбран'. At the bottom of the form is a button labeled 'Загрузить'. The footer text 'Это подвал (footer) страниц сайта' is at the very bottom.

Рис. 8.44. Форма для загрузки аудиофайлов

Как видно из данного рисунка, на форме имеется поле для ввода текста с описанием файла и две кнопки: **Выберите файл** — для выбора файла, **Загрузить** — для загрузки файла в БД сайта.

Если теперь выбрать какой-нибудь аудиофайл (в формате MP3), ввести его описание и нажать на кнопку **Загрузить**, то данный файл будет загружен на сайт, а имя загруженного сайта будет сохранено в БД. После этих действий страница будет выглядеть так, как показано на рис. 8.45.

После этих действий в структуре проекта появится папка для хранения аудиофайлов `media/audios/`, где и будут сохраняться загружаемые файлы, а имена загруженных файлов будут записаны в БД в таблицу `firstapp/audiofile` (рис. 8.46).

**Формы - загрузка аудио файлов**

Описание файла:

Аудио файл:  Выберите файл | Файл не выбран

**Загрузить**

№	Описание файла	Имя файла	Удаление	Прослушать
1	Гаити	audios/gaiti.mp3	<a href="#">Удалить</a>	▶ 0.00 / 0:07

Это подвал (footer) страницы

Рис. 8.45. Форма `form_up_audio.html` после загрузки файла

id	title	file
1	Гаити	audios/gaiti.mp3

Рис. 8.46. Папки для хранения аудиофайлов записи в таблице БД с именами загруженных файлов

Таким образом, мы реализовали загрузку на сайт аудиофайлов, прослушивание и их удаление с сайта.

## 8.9. Краткие итоги

В этой главе были приведены сведения о моделях и о том, как на основе моделей создать таблицы в БД. Мы узнали, как с использованием моделей создавать, читать, редактировать и удалять записи из БД. Выяснили, что через модели можно устанавливать связи между таблицами. Кроме того, мы научились вносить информацию в БД с применением форм. Сейчас у нас есть все необходимые знания для того, чтобы создать свой собственный сайт. Теперь можно перейти к следующей главе, в которой на небольшом и достаточно понятном примере показаны все шаги по созданию сайта. При этом мы больше внимания уделим технологическим вопросам и использованию различных возможностей Django. Однако при этом частично будут рассмотрены и возможности фреймворка Bootstrap для создания привлекательного внешнего вида шаблонов HTML-страниц.



## ГЛАВА 9

# Пример создания веб-сайта на Django

В этой и последующих главах, опираясь на теоретические положения и практические примеры, которые были приведены в книге ранее, мы создадим учебный, но вполне работающий сайт. При этом будут рассмотрены следующие вопросы:

- создание прототипа сайта при помощи Django;
- запуск и остановка сервера для разработки приложения;
- создание модели данных для веб-приложения;
- управление сайтом через административную панель Django.

### ПРИМЕЧАНИЕ

В ходе реализации учебного примера этой главы нужно достаточно внимательно отслеживать, в какой папке (каталоге) приложения требуется размещать те или иные файлы, а также в каком каталоге терминала вы находитесь. Если вы по ошибке создали файл не в том каталоге, который рекомендован, Django не найдет его в нужном месте, и приложение будет выдавать ошибки.

## 9.1. Создание структуры сайта при помощи Django

Создадим сайт с условным названием «Мир книг». Это может быть интернет-ресурс, который дает доступ к скачиванию электронных копий книг, или книжный интернет-магазин, или электронная библиотека, в которую можно записаться и подбирать и заказывать там себе книги в режиме онлайн. Как можно догадаться, цель создания такого сайта заключается в том, чтобы представить онлайн-каталог книг, откуда пользователи смогут загружать доступные книги, и где у них будет возможность управлять своими профилями.

Разработанное нами приложение впоследствии можно будет расширять, улучшать его дизайн, создавать на его основе аналогичные сайты. Но, что более важно, на этом примере вы разберетесь с порядком действий при использовании фреймворка Django для создания веб-приложений.

На первом шаге мы создадим каталог книг, в котором пользователи смогут только просматривать доступные книги. Это позволит нам изучить операции, которые присутствуют при создании практически любого сайта, где осуществляется взаимодействие с базами данных, — чтение и отображение информации из БД.

По мере развития проекта на сайте будут задействованы более сложные функции и возможности Django. Например, мы сможем расширить функционал сайта, позволив пользователям резервировать книги, покажем, как использовать формы для ввода информации в БД, как реализовать авторизацию пользователей и т. п.

Итак, приступим к созданию «скелета» нашего сайта. Запустите приложение PyCharm и создайте новый проект с именем `World_books`. Для этого в главном меню приложения выполните команду **File | New Project**, после чего откроется окно, в котором введите имя нашего проекта: `World_books` (рис. 9.1).



Рис. 9.1. Создание нового проекта `World_books` в окне приложения PyCharm

Для начала обновим базовые библиотеки, которые обеспечивают загрузку необходимых пакетов. Обновим модули `pip` и `setuptools`, запустив в терминале PyCharm следующие команды:

```
easy_install -U pip;
easy_install -U setuptools
```

Теперь загрузим фреймворк Django, выполнив в окне терминала PyCharm команду `pip install Django`.

После этого фреймворк Django появится в списке доступных библиотек (рис. 9.2).

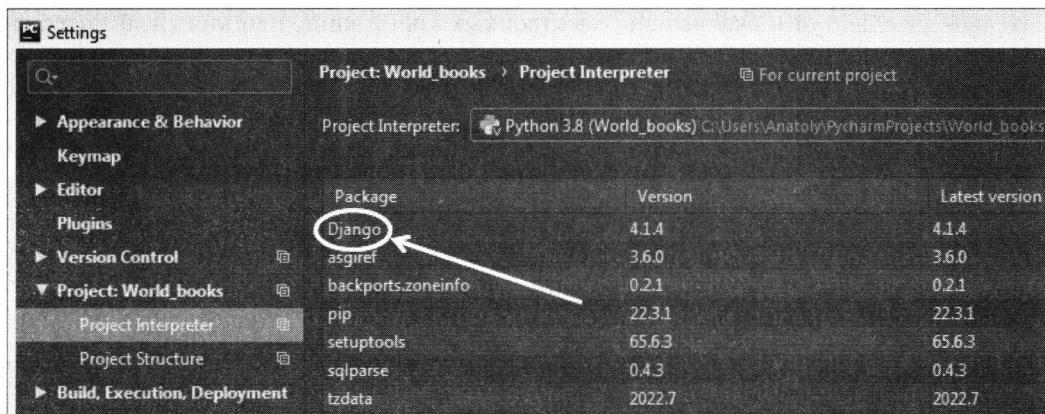
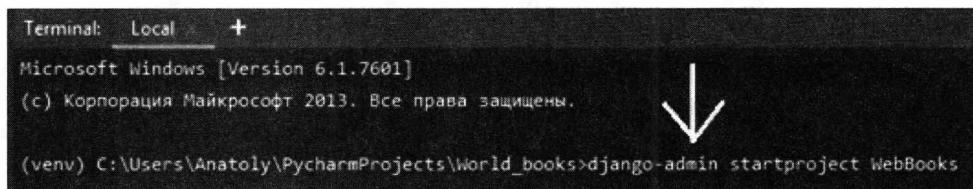


Рис. 9.2. Библиотека Django в списке доступных библиотек проекта `World_books`

Итак, мы создали проект `World_books`, но он появился только в приложении PyCharm и еще не является проектом Django. Поэтому войдите в окно терминала PyCharm и создайте новый проект Django с именем `WebBooks`, выполнив следующую команду (рис. 9.3):

```
django-admin startproject WebBooks
```

В результате выполнения этой команды в папке `World_books` проекта PyCharm будет создана новая папка `WebBooks` проекта Django (рис. 9.4).



```
Terminal: Local +  
Microsoft Windows [Version 6.1.7601]  
(c) Корпорация Майкрософт 2013. Все права защищены.  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>django-admin startproject WebBooks
```

Рис. 9.3. Создание нового проекта Django с именем `WebBooks` в окне терминала PyCharm

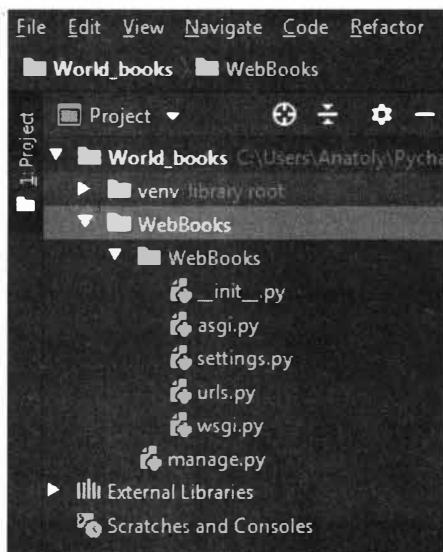


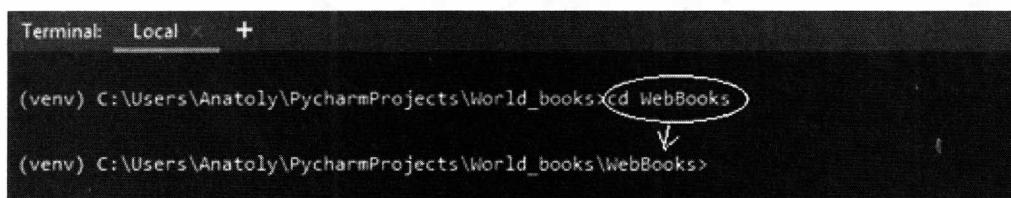
Рис. 9.4. Структура файлов проекта Django с именем `WebBooks`

Как видно из рис. 9.4, головная папка проекта `WebBooks` включает одну вложенную папку `WebBooks` — это ключевой каталог нашего проекта, в котором находятся следующие файлы:

- `settings.py` — содержит все настройки проекта (здесь мы регистрируем приложения, задаем размещение статичных файлов, настраиваем базу данных и т. п.);
- `urls.py` — задает ассоциации интернет-адресов (URL) с представлениями `views` (этот файл может содержать все настройки URL, но обычно его делят на части — по одной на каждое приложение, как будет показано далее);
- `asgi.py` и `wsgi.py` — предназначены для организации связи между Django-приложением и внешним веб-сервером (мы задействуем эти файлы позже, когда будем рассматривать развертывание сайта в Интернете).

Кроме вложенной папки `WebBooks`, в головной папке проекта `WebBooks` имеется также файл `manage.py`. Это скрипт, который обеспечивает создание приложений, работу с базами данных, запуск отладочного сервера. Именно этот скрипт потребуется на следующем шаге для создания приложения.

Итак, создадим приложение с именем `catalog`. Для этого сначала необходимо из приложения PyCharm (внешняя папка `World_books`) перейти в приложение Django — войти во вложенную папку `WebBooks`, в которой находится скрипт `manage.py`. Для этого в окне терминала PyCharm выполните команду `cd WebBooks` (рис. 9.5).

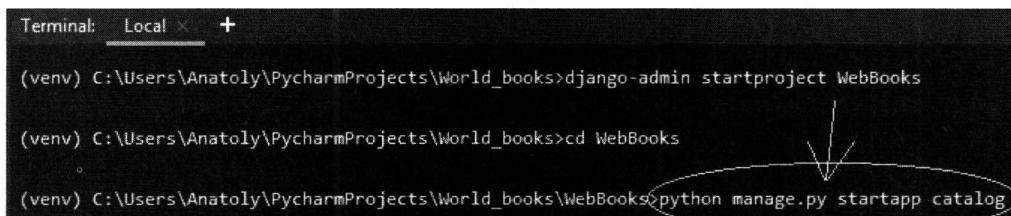


```
Terminal: Local +  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>cd WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books\WebBooks>
```

Рис. 9.5. Переход в папку с именем `WebBooks` проекта Django

Вот теперь, находясь в папке проекта Django с именем `WebBooks` (той, где размещен скрипт `manage.py`), создадим приложение `catalog`. Для этого выполним следующую команду (рис. 9.6):

```
python manage.py startapp catalog
```



```
Terminal: Local +  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>django-admin startproject WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books>cd WebBooks  
(venv) C:\Users\Anatoly\PycharmProjects\World_books\WebBooks>python manage.py startapp catalog
```

Рис. 9.6. Создание приложения `catalog` в проекте Django с именем `WebBooks`

В результате в папке верхнего уровня `WebBooks` появится новая папка с именем `catalog`, а в ней будут содержаться файлы нашего приложения `catalog` (рис. 9.7).

Эти файлы предназначены для выполнения следующих функций:

- папка `migrations` — хранит миграции (файлы, которые позволяют автоматически обновлять базу данных по мере изменения моделей данных);
- «пустой» файл `__init__.py` — его присутствие позволяет Django и Python распознавать папку `catalog` как папку с модулями Python, что дает возможность использовать объекты этих модулей внутри других частей проекта;
- файл `admin.py` — содержит информацию с настройками административной части приложения;
- файл `apps.py` — предназначен для регистрации приложений;
- файл `models.py` — содержит описание моделей данных, с которыми будет работать приложение;

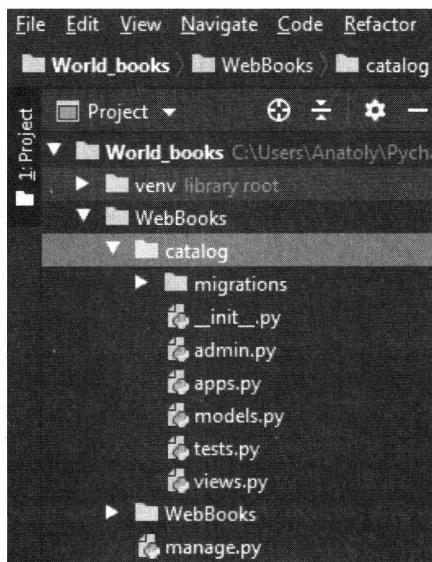


Рис. 9.7. Содержимое папки catalog

- файл `tests.py` — содержит тесты для тестирования приложения;
- файл `views.py` — содержит контроллеры или представления (`views`), которые обеспечивают взаимодействие приложения с БД (выборка, добавление, удаление записей) и с запросами пользователя.

Большинство этих файлов уже содержат некоторый шаблонный программный код для работы с указанными объектами. Однако следует отметить, что для нашего проекта некоторых необходимых папок и файлов не хватает. В частности, не были созданы папки для файлов шаблонов и статичных файлов. Далее мы создадим эти папки и файлы (они не обязательны для каждого сайта, но будут нужны в нашем примере).

Созданное приложение нужно зарегистрировать в проекте Django. Это необходимо для того, чтобы различные утилиты распознавали его при выполнении некоторых действий (например, при добавлении моделей в базу данных). Приложения регистрируются добавлением их названий в список `INSTALLED_APPS` в файле настроек проекта `settings.py`.

Откройте файл `WebBooks\WebBooks\settings.py`, найдите в нем список зарегистрированных приложений `INSTALLED_APPS` и добавьте наше приложение `catalog` в конец списка, как показано в листинге 9.1 (выделено серым фоном и полужирным шрифтом) и на рис. 9.8.

#### Листинг 9.1. Изменения в файле `settings.py`

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

The screenshot shows the PyCharm IDE interface. On the left is the Project tool window displaying the directory structure of the Django project. The main editor window shows the `settings.py` file. A red arrow points from the `catalog` application in the project tree to the line of code where it is registered in the `INSTALLED_APPS` list.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books > WebBooks > WebBooks > settings.py
Project  settings.py
World_books C:\Users\Ana>
  venv library root
  WebBooks
    catalog
    WebBooks
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
  manage.py
  32
  33 INSTALLED_APPS = [
  34     'django.contrib.admin',
  35     'django.contrib.auth',
  36     'django.contrib.contenttypes',
  37     'django.contrib.sessions',
  38     'django.contrib.messages',
  39     'django.contrib.staticfiles',
  40     'catalog',
  41 ]
  42

```

Рис. 9.8. Регистрация приложения catalog в проекте Django

Теперь следует указать базу данных для нашего проекта. По возможности имеет смысл использовать одну и ту же базу данных, как в процессе разработки проекта, так и при размещении его в Интернете. Это позволит исключить возможные различия в поведении проекта после его публикации.

Однако для нашего проекта мы воспользуемся базой данных SQLite, потому что по умолчанию приложение уже настроено для работы с ней. В этом можно убедиться, открыв соответствующий раздел в файле `settings.py` (рис. 9.9).

The screenshot shows the PyCharm IDE interface. The Project tool window is visible on the left. The main editor window shows the `settings.py` file. A red arrow points from the `catalog` application in the project tree to the `DATABASES` configuration block in the code.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books > WebBooks > WebBooks > settings.py
Project  settings.py
World_books C:\Users\Ana>
  venv library root
  WebBooks
    catalog
    WebBooks
      __init__.py
      asgi.py
      settings.py
      urls.py
      wsgi.py
  manage.py
  74 # Database
  75 # https://docs.djangoproject.com/en/3.0/ref/settings/#i
  76
  77 DATABASES = {
  78     'default': {
  79         'ENGINE': 'django.db.backends.sqlite3',
  80         'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
  81     }
  82 }
  83
  84
  85 # Password validation

```

Рис. 9.9. Настройки проекта для работы с базой данных SQLite

Как можно видеть, к проекту подключен движок БД SQLite, а файл, который содержит сами данные, носит имя `db.sqlite3`.

Файл `settings.py` также служит и для выполнения некоторых других настроек (значения их по умолчанию показаны на рис. 9.10):

- LANGUAGE\_CODE — строка, представляющая код языка. Значение по умолчанию: en-us — американская версия английского. Код русского языка для России — ru-ru;
- TIME\_ZONE — строка, представляющая часовой пояс. Значение по умолчанию: UTC (America/Chicago). Значение для России: Europe/Moscow;
- USE\_I18N — логическое значение, которое указывает, должна ли быть включена система перевода Django. По умолчанию: True. Если значение USE\_I18N установить в False, то Django не будет загружать механизм перевода — это один из способов повышения производительности приложения;
- USE\_L10N — логическое значение, которое указывает, будет ли включено локализованное форматирование данных. По умолчанию: True (при этом Django будет отображать числа и даты в формате текущей локализации);
- USE\_TZ — логическое значение, которое указывает, будет ли datetime привязан к часовому поясу. По умолчанию True (при этом Django будет использовать внутренние часы с учетом часового пояса, в противном случае будут задействованы нативные даты по местному времени).

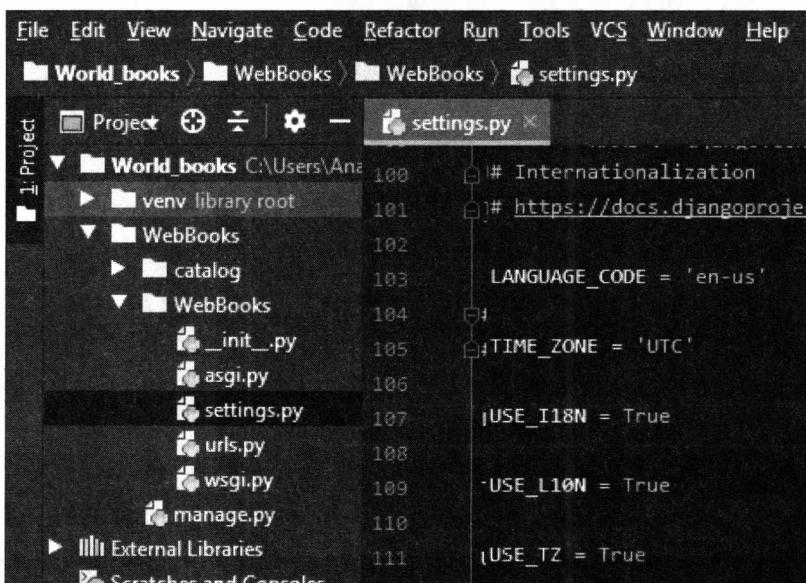


Рис. 9.10. Настройки проекта в файле settings.py по умолчанию

В нашем случае имеет смысл поменять параметры LANGUAGE\_CODE и TIME\_ZONE. В привязке этих параметров к русскому языку и Московскому региону эти настройки будут выглядеть как в листинге 9.2 (изменения выделены серым фоном).

#### Листинг 9.2. Изменения в файле settings.py

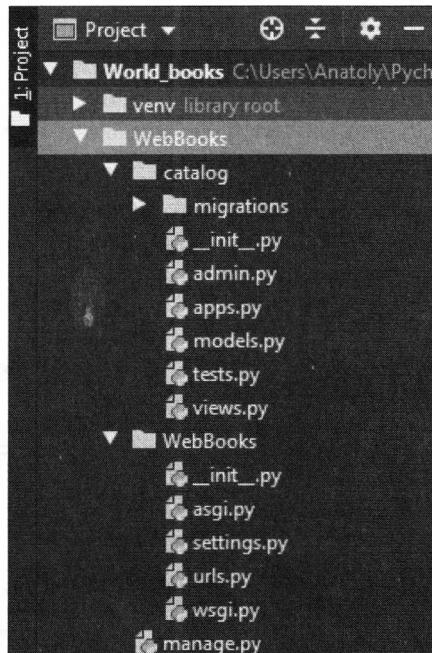
```

LANGUAGE_CODE = 'ru-ru'
TIME_ZONE = 'Europe/Moscow'
USE_I18N = True

```

```
USE_L10N = True
USE_TZ = True
```

В проекте может быть несколько приложений, но мы ограничимся одним. И после всех выполненных нами действий полная структура нашего веб-приложения будет выглядеть так, как показано на рис. 9.11.



**Рис. 9.11.** Полная структура файлов и папок сайта с проектом WebBook и приложением catalog

Теперь определим какие-нибудь простейшие действия, которые будет выполнять это приложение, например, отправлять в ответ пользователю строку **Главная страница сайта "Мир книг"**.

Для этого перейдем в проекте приложения catalog к файлу catalog/views.py и напишем в нем код листинга 9.3.

#### Листинг 9.3. Код файла views.py

```
from django.shortcuts import render
from django.http import HttpResponse
def index(request):
    return HttpResponse("Главная страница сайта " "Мир книг" !")
```

Здесь мы сначала импортируем класс `HttpResponse()` из стандартного пакета `django.http`. Затем определяем функцию `index()`, которая в качестве параметра получает объект запроса пользователя `request`. Класс `HttpResponse()` предназначен для создания ответа, который отправляется пользователю. Так что с помощью выражения `return HttpResponse()` мы как раз и отправляем пользователю строку "Главная страница сайта "Мир книг"!".

Теперь в проекте Django откроем файл urls.py, обеспечивающий сопоставление маршрутов с представлениями (views), которые будут обрабатывать запросы по этим маршрутам. По умолчанию этот файл выглядит так:

```
from django.contrib import admin
from django.urls import path
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

В первой строке из модуля django.contrib импортируется класс admin, который предоставляет возможности работы с интерфейсом администратора сайта. Во второй строке из модуля django.urls импортируется функция path. Она задает возможность сопоставлять запросы пользователя (определенные маршруты) с функциями их обработки. Так, в нашем случае маршрут 'admin/' будет обрабатываться методом admin.site.urls. Если пользователь запросит показать страницу администратора сайта ('admin/'), то будет вызван метод admin.site.urls, который вернет пользователю HTML-страницу администратора.

Но ранее мы определили функцию index() в файле views.py, который возвращает пользователю текстовую строку. Поэтому изменим сейчас файл urls.py, как показано в листинге 9.4 (изменения выделены серым фоном).

#### Листинг 9.4. Измененный код модуля urls.py

```
from django.contrib import admin
from django.urls import path
from catalog import views
urlpatterns = [
    path('', views.index, name='index'),
    path('admin/', admin.site.urls),
]
```

Чтобы использовать функцию views.index, мы здесь сначала импортируем модуль views. Затем сопоставляем маршрут ('') — это, по сути, запрос пользователя к корневой (главной) странице сайта, с функцией views.index. Если пользователь запросит показать главную страницу сайта (''), то будет вызвана функция index из файла views.py. А в этой функции мы указали, что пользователю нужно вернуть HTML-страницу с единственным сообщением: **Главная страница сайта "Мир книг"!**. Соответственно, и маршрут с именем 'index' также будет сопоставляться с запросом к «корню» приложения.

Теперь запустим приложение командой

```
python manage.py runserver
```

и перейдем в браузере по адресу <http://127.0.0.1:8000/> — браузер отобразит нам сообщение: **Главная страница сайта "Мир книг"!**! (рис. 9.12).

Вспомним, что в файле urls.py (см. листинг 9.4) второй строкой прописана возможность запуска встроенного в Django приложения «Администратор сайта» — через маршрут 'admin/'. Вызовем этот модуль, указав в браузере интернет-адрес (URL) <http://127.0.0.1:8000/admin/>, и получим следующий ответ (рис. 9.13).

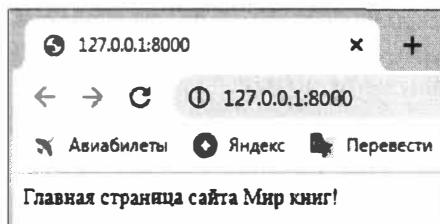


Рис. 9.12. Первый запуск проекта WebBook с приложением catalog

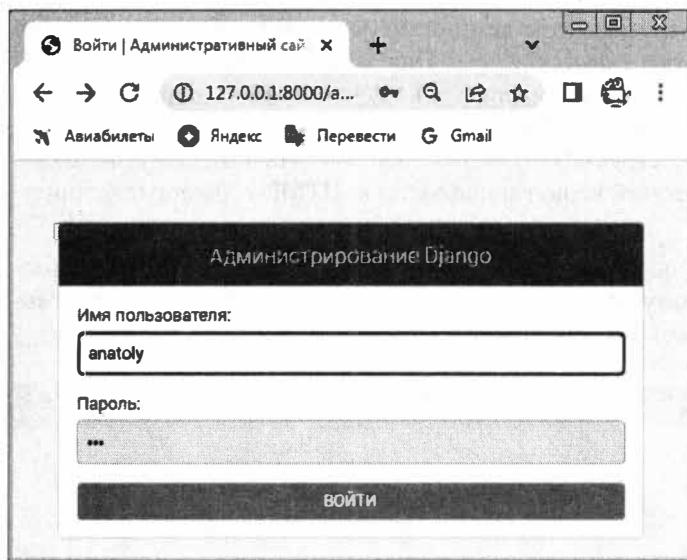


Рис. 9.13. Вызов административной страницы сайта проекта WebBook

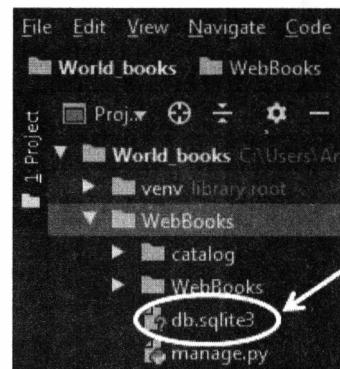


Рис. 9.14. База данных db.sqlite3

Как можно видеть, интерфейс приложения администрирования сайта выведен на русском языке. Это результат того, что в файле `settings.py` мы изменили параметры локализации и установили русский язык (параметр `ru-ru`). Обратите внимание, что после вызова административной панели в каталоге нашего проекта появился файл с базой данных `db.sqlite3` (рис. 9.14).

К приложению «Администратор сайта» мы вернемся немного позже. А пока можно констатировать, что мы создали «скелет» веб-приложения, которое теперь можно расширять, добавляя новые страницы, представления (`views`), модели (`models`) и устанавливая URL-соответствия между представлениями и моделями данных. Самое время перейти к следующему разделу и начать создавать базу данных, а также писать код, который "научит" сайт делать то, что он должен делать.

## 9.2. Установка дополнительных пакетов и настройка параметров сайта «Мир книг»

Для функционирования сайта потребуется установить несколько дополнительных пакетов и подключить их к проекту. Нам понадобятся следующие пакеты:

- Pillow — библиотека для работы с изображениями;
- django\_cleanup — для очистки хранилища загруженных медиафайлов (изображения, видео, аудио и т. п.).

Кроме того, необходимо создать папку для хранения шаблонов HTML-страниц, в настройках проекта указать, где будут находиться файлы следующих типов:

- статические файлы (static), которые загружаются разработчиками на этапе проектирования сайта;
- шаблоны HTML-страниц (templates);
- медиафайлы (media — изображения, PDF-файлы, документы и т. п.), которые будут загружены пользователями в процессе работы с сайтом.

Начнем с загрузки дополнительных пакетов.

Так как для макетирования страниц сайта мы будем использовать фреймворк Bootstrap, то его необходимо подключить к нашему проекту. В предыдущей главе мы файлы этого фреймворка скачивали с официального сайта и добавляли их в папку static, здесь поступим таким же образом.

Для начала нужно загрузить пакет Pillow, подав команду в терминале PyCharm:

```
pip install Pillow
```

Для очистки папок с медиаданными от удаленных файлов нам понадобится пакет django-cleanup. В этом пакете реализована автоматическая очистка папок с удаленными медиафайлами, что избавляет программистов от написания соответствующего программного кода. Для загрузки пакета очистки файловых хранилищ выполним команду

```
pip install django-cleanup
```

После этого можно проверить, что эти пакеты действительно загружены в проект (рис. 9.15).

После этого пакет django-cleanup нужно подключить к проекту. Открываем файл WebBooks/WebBooks/settings.py и в блок INSTALLED\_APPS добавляем строку из листинга 9.5 (изменения выделены серым фоном).

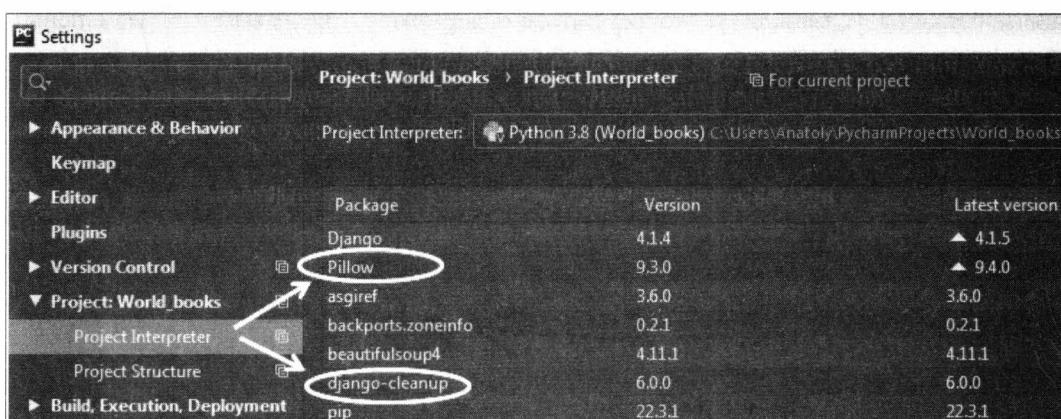


Рис. 9.15. Результаты загрузки пакетов Pillow и django-cleanup

### Листинг 9.5. Изменения в файле settings.py

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

Теперь в файле настроек `WebBooks/WebBooks/settings.py` нужно указать места расположения статических файлов и медиафайлов (листинг 9.6, изменения выделены серым фоном).

### Листинг 9.6. Изменения в файле settings.py

```
STATIC_URL = 'static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR / "static"),
]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

Здесь определена папка, в которой будут храниться статические файлы (`static`). В строке (`MEDIA_URL`) задан URL-адрес, в строке (`MEDIA_ROOT`) — путь к папке `media`, в которой будут храниться медиафайлы пользователей. Если в моделях мы укажем имя папки для хранения изображений `upload_to='images'`, а здесь в настройках задали папку для хранения медиафайлов `media`, то в нашем случае все изображения будут размещены по следующему пути: `/media/images/`.

Нам не нужно создавать папки `media` и `images`, Django создаст их автоматически, как только пользователь выполнит процедуру загрузки первого изображения. А вот папки `WebBooks/static` и `WebBooks/templates` нужно будет создать самостоятельно. Создадим две новые папки (`static, templates`) и скорректируем содержимое модуля `settings.py`. Изменения, внесенные в модуль `settings.py`, показаны на рис. 9.16.

Теперь в настройках нашего проекта нужно выполнить еще одно действие — указать расположение папки `templates` с шаблонами HTML-страниц. Снова открываем модуль `WebBooks/WebBooks/settings.py` и добавляем в него код листинга 9.7 (выделено серым фоном), показано на рис. 9.17.

### Листинг 9.7. Изменения в файле settings.py

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR, ],
        'APP_DIRS': True,
```

The screenshot shows the PyCharm IDE interface. The project tree on the left shows a structure with 'World\_books' at the root, containing 'venv', 'library root', and 'WebBooks'. 'WebBooks' contains 'catalog', 'media', 'static', 'templates', and 'WebBooks' (which further contains '\_init\_.py', 'asgi.py', 'settings.py', 'urls.py', and 'wsgi.py'). Other files like 'db.sqlite3' and 'manage.py' are also listed. The 'External Libraries' section is shown at the bottom. The main code editor window displays 'settings.py' with code related to static files and media URLs. An arrow points from the 'Project' tool window to the 'settings.py' file in the code editor.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books > WebBooks > WebBooks > settings.py
Project  settings.py index.html base.html admin.py
World_books C:\Users\Ana
  venv library root
  WebBooks
    catalog
    media
      static
    templates
    WebBooks
      __init__.py
      asgi.py
      settings.py ->
      urls.py
      wsgi.py
      db.sqlite3
      manage.py
External Libraries

```

```

114
115     USE_TZ = True
116
117     .
118     # Static files (CSS, JavaScript, Images)
119     # https://docs.djangoproject.com/en/4.1/howto/static-files/
120
121     STATIC_URL = 'static/'
122
123     STATICFILES_DIRS = [
124         os.path.join(BASE_DIR / "static"),
125     ]
126
127     MEDIA_URL = '/media/'
128     MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
129

```

Рис. 9.16. Указание путей к статическим файлам и медиафайлам в настройках проекта

The screenshot shows the PyCharm IDE interface. The project tree on the left shows a structure with 'World\_books' at the root, containing 'venv', 'library root', and 'WebBooks'. 'WebBooks' contains 'catalog', 'media', 'static', 'templates', and 'WebBooks' (which further contains '\_init\_.py', 'asgi.py', 'settings.py', 'urls.py', and 'wsgi.py'). Other files like 'db.sqlite3' and 'manage.py' are also listed. The 'Scratches and Consoles' section is shown at the bottom. The main code editor window displays 'settings.py' with code related to template configurations. Two arrows point from the 'Project' tool window to the 'TEMPLATES' and 'OPTIONS' sections in the code editor.

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books > WebBooks > WebBooks > settings.py
Project  settings.py index.html base.html admin.py WebBooks.urls.py
World_books C:\Users\Ana
  venv library root
  WebBooks
    catalog
    media
    static
    templates
    WebBooks
      __init__.py
      asgi.py
      settings.py ->
      urls.py
      wsgi.py
      db.sqlite3
      manage.py
Scratches and Consoles

```

```

55     ROOT_URLCONF = 'WebBooks.urls'
56
57     TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
58
59     TEMPLATES = [
60         {
61             'BACKEND': 'django.template.backends.django.DjangoTemplates',
62             'DIRS': [TEMPLATE_DIR, ],
63             'APP_DIRS': True,
64             'OPTIONS': {
65                 'context_processors': [
66                     'django.template.context_processors.debug',
67                     'django.template.context_processors.request',
68                     'django.contrib.auth.context_processors.auth',
69                     'django.contrib.messages.context_processors.messages',
70                 ],
71             },
72         },
73     ]

```

Рис. 9.17. Указание путей к шаблонам HTML-страниц в настройках проекта

```
'OPTIONS': {
    'context_processors': [
        'django.template.context_processors.debug',
        'django.template.context_processors.request',
        'django.contrib.auth.context_processors.auth',
        'django.contrib.messages.context_processors.messages',
    ],
},
},
],
}
```

Теперь нужно скачать файлы Bootstrap и загрузить их в папку static нашего проекта. Скачиваем готовые файлы с официального сайта по следующей ссылке:

<https://getbootstrap.com/docs/5.2/getting-started/download/>.

После скачивания вы получите архивированный zip-файл, в имени которого будет отражена текущая версия фреймворка, например bootstrap-5.2.3-dist.zip. После распаковки архива будут созданы две папки: css — с файлами стилей, и js — с Java-скриптами. Эти две папки нужно перенести в ваш проект в каталог static, там же создадим пустую папку images, где будут храниться статичные изображения, загружаемые на этапе проектирования сайта (рис. 9.18).

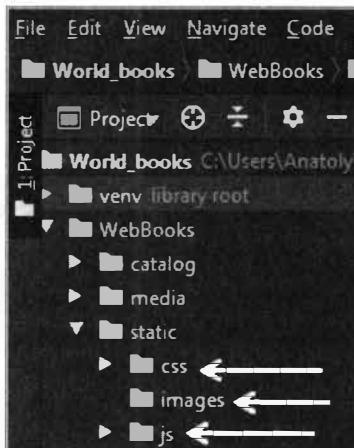


Рис. 9.18. Загрузка файлов Bootstrap 5 в папку static проекта

Итак, мы выполнили основные шаги по инициализации нашего проекта, теперь можно приступить к разработке структуры моделей данных сайта.

## 9.3. Разработка структуры моделей данных сайта «Мир книг»

В этом разделе мы узнаем, как строить модели для сайта «Мир книг», и рассмотрим некоторые из основных типов полей таблиц, в которых будет храниться информация сайта. Здесь также будет показано несколько основных способов доступа к данным этих моделей.

Веб-приложения Django получают доступ и управляют данными через объекты Python, называемые *моделями*. Модели определяют структуру хранимых данных, включая типы полей, их максимальный размер, значения по умолчанию, параметры выбора, текст меток для форм и пр. Структура и описание модели не зависят от используемой базы данных — она может быть выбрана позднее путем изменения всего нескольких настроек проекта. После выбора какой-либо базы данных разработчику не придется напрямую взаимодействовать с ней (создавать таблицы и связи между ними) — это сделает Django, используя уже разработанную структуру модели.

Перед тем как перейти к созданию моделей данных, нам необходимо определить, какую информацию мы будем хранить в БД, сформировать перечень таблиц БД и организовать связи между ними.

Итак, на нашем сайте мы будем хранить следующие данные о книгах:

- название книги;
- автор или авторы книги;
- год издания;
- издательство;
- краткое описание книги (аннотация);
- язык, на котором написана книга;
- жанр книги;
- ISBN (International Standard Book Number, международный стандартный книжный номер).

Кроме того, необходимо хранить сведения о количестве доступных экземпляров книги, о статусе доступности каждого экземпляра и т. п. Нам также понадобится иметь более подробную информацию об авторе (авторах) книги, чем просто его фамилия и имя, ведь у книги может быть несколько авторов с одинаковыми или похожими именами. Конечно, потребуется реализовать возможность сортировки информации на основе названия книги, автора, языка, жанра и других признаков.

При проектировании моделей данных имеет смысл создавать самостоятельные модели для каждого объекта. Под *объектом* мы будем понимать группу связанный информации. В нашем случае очевидными объектами являются: жанры книг, сами книги, авторы, издательства, экземпляры книг, языки.

Следует также определить, какие данные можно и желательно выдавать в виде списка выбора. Это рекомендуется в тех случаях, когда все возможные варианты списка заранее известны или относительно постоянны, но могут впоследствии измениться. Можно выделить следующие данные, которые можно выдавать в виде списков:

- жанр книги (фантастика, поэзия);
- язык (русский, английский, французский, японский);
- издательство.

Как только будет определен перечень моделей данных (таблицы БД и поля таблиц), нужно подумать об отношениях между моделями (между таблицами в БД). Как было

показано в предыдущей главе, Django позволяет определять отношения «один к одному» (конструктор `OneToOneField`), «один ко многим» (конструктор `ForeignKey`) и «многие ко многим» (конструктор `ManyToManyField`).

Для нашего сайта мы предусмотрим следующие таблицы для хранения информации, и в соответствии с этим нам потребуется сформировать структуру и описание моделей данных. Вот их перечень:

- книги (общие сведения о книгах);
- экземпляры книг (статус конкретных физических книг, доступных в системе);
- автор (авторы) книги;
- издательства;
- язык, на котором написана книга;
- жанры книг.

Структура этих моделей данных показана на рис. 9.19.

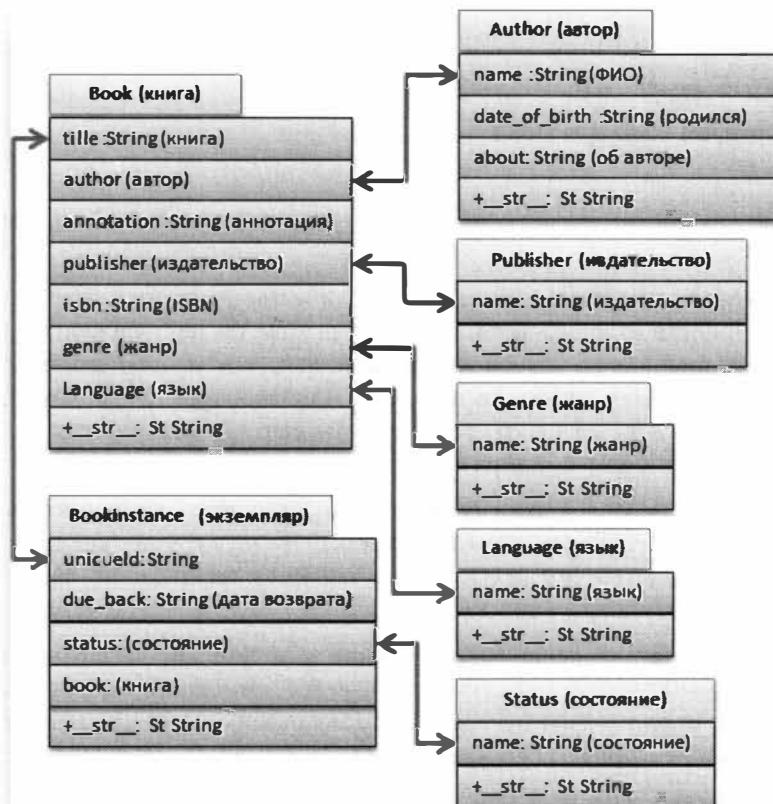


Рис. 9.19. Структура моделей данных сайта «Мир книг»

Из данного рисунка видно, что мы сформировали перечень таблиц, которые будут созданы в БД, перечень и типы полей, которые будут в каждой таблице, определили связи между таблицами.

Итак, мы определились с перечнем хранимых данных. На основе этой информации можно приступить к созданию моделей данных. Однако прежде познакомимся с основными элементами, которые нужно будет создавать при формировании моделей данных. Этому и посвящен следующий раздел.

## 9.4. Основные элементы моделей данных в Django

В этом разделе мы разберемся с тем, как формируются модели данных, рассмотрим некоторые из наиболее важных полей и их аргументы, узнаем, что такое метаданные и как с помощью методов можно управлять данными в моделях.

Модели в приложении обычно описываются в файле `models.py`. Они реализуются как подклассы или объекты класса `django.db.models.Model` и могут включать поля, метаданные и методы. В приведенном далее фрагменте кода показан пример типичной модели, имеющей условное название `MyModelName`:

```
from django.db import models

# Типичный класс, определяющий модель, производный от класса Model
class MyModelName(models.Model):
    # Поле (или множество полей)
    my_field_name = models.CharField(max_length=20,
                                      help_text="Не более 20 символов")
                                      verbose_name="Ведите ФИО")

    # Метаданные
    class Meta:
        ordering = ["-my_field_name"]

    # Методы
    def get_absolute_url(self):
        # Возвращает url-адрес для доступа к экземпляру MyModelName
        return reverse('model-detail-view', args=[str(self.id)])
    def __str__(self):
        # Стока для представления объекта MyModelName в Admin site
        return self.field_name
```

В этом программном коде определена модель данных (будущая таблица в БД) с именем MyModelName. Таблица БД будет иметь поле my\_field\_name для хранения данных (с числом символов в имени не более 20). Предусмотрена сортировка данных по полю my\_field\_name и в виде функций заданы два метода. В следующих разделах мы более подробно рассмотрим каждый из элементов внутри модели.

#### **9.4.1. Поля и их аргументы в моделях данных**

Модель может иметь произвольное число полей любого типа. Каждое поле представляется столбец данных, который будет храниться в одной из таблиц базы данных. Каждая запись (строка в таблице базы данных) будет состоять из значений тех полей, которые описаны в модели. Давайте рассмотрим описание поля из примера, приведенного в предыдущем разделе:

В соответствии с этим описанием таблица БД будет иметь одно поле с именем `my_field_name`, тип поля — `CharField`. Этот тип поля позволяет хранить в БД строки, состоящие из буквенных и цифровых символов. Система предусматривает проверку содержания этого поля при вводе в него пользователем каких-либо значений в формате HTML. Django не позволит ввести в него недопустимые символы.

Поля могут принимать некоторые аргументы, которые дополнительно определяют, как поле будет хранить данные или как оно может применяться пользователем. В нашем примере поле имеет три аргумента:

- `max_length=20` — указывает, что максимальная длина этого поля составляет 20 символов;
- `help_text="Не более 20 символов"` — предоставляет дополнительную текстовую подсказку, чтобы помочь пользователям узнать, какое ограничение наложено на длину этого поля;
- `verbose_name="Введите ФИО"` — текстовая метка, которая будет выведена в форме рядом с этим полем.

Метка поля и подсказки будут показаны пользователю через HTML-форму.

Имя поля используется для обращения к нему в запросах и шаблонах. У каждого поля также есть *метка*, значение которой можно задать через аргумент `verbose_name`. Метка по умолчанию задается автоматически путем изменения первой буквы имени поля на заглавную букву и замены любых символов подчеркивания пробелом. Например, если аргумент `verbose_name` не будет задан, то поле с именем `my_field_name` будет иметь метку по умолчанию «`my field name`». В нашем случае, поскольку аргумент `verbose_name` определен, поле будет иметь текстовую метку «**Введите ФИО**».

Поля по умолчанию отображаются в форме (например, на сайте администратора) в том же порядке, в котором они объявляются, хотя этот порядок можно переопределить в процессе разработки проекта.

Поля, описываемые в моделях данных и имеющие разные типы, могут иметь достаточно большое количество *общих аргументов*:

- `help_text` — предоставляет текстовую метку для HTML-форм (например, на сайте администратора или на форме пользователя);
- `verbose_name` — удобно читаемое имя для поля, используемое в качестве его метки. Если этот аргумент не указан, то Django автоматически сформирует его по умолчанию от имени поля;
- `default` — значение по умолчанию для поля. Это может быть значение или вызываемый объект, и в этом случае объект будет вызываться каждый раз, когда создается новая запись;
- `null` — если аргумент имеет значение `True`, то Django будет хранить пустые значения в базе данных для полей как `NULL`, где это уместно (поле типа `CharField` вместо этого сохранит пустую строку). По умолчанию принимается значение `False`;
- `blank` — если аргумент имеет значение `True`, поле в ваших формах может быть пустым. По умолчанию принимается значение `False`, означающее, что проверка формы Django заставит вас ввести в это поле некоторое значение. Этот аргумент часто ис-

пользуется с аргументом `null = True` — если вы разрешаете ввод пустых значений, то также должны обеспечить, чтобы база данных могла принять эти пустые значения;

- `choices` — предоставление выбора из вариантов значений для этого поля. Если параметр `choices` задан, то соответствующий виджет формы будет являться полем с вариантами выбора из нескольких значений (вместо стандартного текстового поля);
- `primary_key` — если аргумент имеет значение `True`, то текущее поле будет выступать в качестве *первичного ключа* для модели (первичный ключ — это специальный столбец базы данных, предназначенный для однозначной идентификации всех разных записей таблицы). Если в качестве первичного ключа не указано ни одно поле, то Django автоматически добавит ключевое поле (`id`) в таблицу данных.

Указанные общие аргументы могут использоваться при объявлении полей следующих типов:

- `CharField` — служит для определения строк фиксированной длины: от короткой до средней. Следует указывать максимальную длину строки `max_length` для хранения данных;
- `TextField` — предназначено для больших строк произвольной длины. Вы можете указать для поля `max_length`, но это значение будет задействовано только тогда, когда поле отображается в формах (оно не применяется на уровне базы данных);
- `IntegerField` — это поле для хранения значений (целого числа) и для проверки введенных в форму значений в виде целых чисел;
- `DateField` и `DateTimeField` — служат для хранения дат и информации о дате или времени (как в Python `datetime.date` и `datetime.datetime` соответственно). Эти поля могут дополнительно объявлять параметры:
  - `auto_now=True` (для установки поля на текущую дату каждый раз, когда модель сохраняется);
  - `auto_now_add` (только для установки даты, когда модель была впервые создана);
  - по умолчанию (чтобы установить дату по умолчанию, которую пользователь может переустановить);
- `EmailField` — служит для хранения и проверки адресов электронной почты;
- `FileField` и `ImageField` — предназначены для загрузки файлов и изображений (`ImageField` просто добавляет дополнительную проверку, является ли загруженный файл изображением). Они имеют параметры для определения того, как и где хранятся загруженные файлы;
- `AutoField` — это особый тип `IntegerField`, который автоматически увеличивается. Первичный ключ (`id`) этого типа автоматически добавляется в вашу модель, если вы явно не укажете его;
- `ForeignKey` — служит для указания отношения «один ко многим» к другой модели базы данных (например, автомобиль определенной модели имеет одного производителя, но некоторый производитель может делать много автомобилей разных марок). Сторона отношения «один» — это модель, содержащая данный ключ;
- `ManyToManyField` — служит для определения отношения «многие ко многим» (например, книга может иметь несколько жанров, и каждый жанр может содержать не-

сколько книг). В нашем приложении для сайта «Мир книг» мы будем использовать этот тип поля аналогично типу `ForeignKey` для описания отношений между группами. Эти типы полей имеют параметр `on_delete`, определяющий, что происходит, когда связанная запись удаляется (например, значение `models.SET_NULL` просто удаленно установило бы значение поля в `NULL`).

Существует много других типов полей, включая поля для разных типов чисел (большие целые числа, малые целые числа, дробные). А также поля с логическими значениями, URL-адресами, символами «slugs», уникальными идентификаторами, временем, датами и т. д. Эти поля мы достаточно подробно описали, когда рассматривали формы Django.

## 9.4.2. Метаданные в моделях Django

*Метаданные* — это информация о другой информации или данные, относящиеся к дополнительным сведениям о содержимом или объекте. Метаданные раскрывают сведения о признаках и свойствах, характеризующих какие-либо сущности, позволяющие автоматически искать и управлять ими в больших информационных потоках. К метаданным относятся многие параметры модели, заданные по умолчанию, которые, кстати, можно переопределить. Это, например, имена таблиц, порядок сортировки данных, список индексов, имя поля и т. п.

В Django имеется возможность объявить метаданные о своей модели на уровне самой модели. Для этого нужно объявить класс `Meta` в теле класса модели:

```
class Meta:  
    ordering = ["-my field name"]
```

В этом коде показана одна из наиболее полезных функций метаданных — управление сортировкой записей. Порядок сортировки можно задать, указав имя поля (или полей). Сам порядок сортировки зависит от типа поля. Например, для текстового поля сортировка будет выполняться в алфавитном порядке, а поля даты будут отсортированы в хронологическом порядке. Прямой порядок сортировки можно заменить на обратный, для этого добавляют символ "минус" (-). В этом примере данные текстового поля `my_field_name` (имя клиента) будут отсортированы в обратном алфавитном порядке: от буквы «Я» до буквы «А», поскольку здесь присутствует символ (-).

Если, например, мы решили по умолчанию сортировать книги по двум полям и написали следующий код:

```
class Meta:  
    ordering = ["title", "-pubdate"]
```

то книги будут отсортированы по названию согласно алфавиту от А до Я, а затем по дате публикации внутри каждого названия — от самого нового (последнего) издания до самого старого (первого).

Другим распространенным атрибутом является `verbose_name`, или подробное (многословное) имя для класса. Например:

```
class Meta:  
    verbose_name = "Название книги"
```

#### ПРИМЕЧАНИЕ

Полный список метаданных доступен в документации по Django.

### 9.4.3. Методы в моделях Django

Модель также может иметь *методы*, т. е. различные способы манипулирования данными (чтение, обновление, удаление). Минимально в каждой модели должен быть определен стандартный метод класса для Python: `__str__()`. Это необходимо для того, чтобы вернуть удобно читаемую строку для каждого объекта. Такая строка используется для представления отдельных записей в модуле администрирования сайта (и в любом другом месте, где вам нужно обратиться к экземпляру модели). Часто этот метод возвращает наименование поля из модели, например:

```
def __str__(self):  
    return self.field_name
```

Если необходимо вернуть несколько полей, то потребуется другая инструкция:

```
def __str__(self):  
    return '%s, %s' % (self.field_name_1, self.field_name_2)
```

Еще один распространенный метод, который включается в модели Django, — метод `get_absolute_url()`, который возвращает URL-адрес для отображения отдельных записей модели на веб-сайте. Если этот метод определен, то Django автоматически добавит кнопку **Просмотр на сайте** на экранах редактирования записей модели (на сайте администратора). Вот типичный шаблон для `get_absolute_url()`:

```
def get_absolute_url(self):  
    return reverse('model-detail-view', args=[str(self.id)])
```

Предположим, что требуется использовать URL-адрес, например:

**/myapplication/mymodelname/2**

для отображения отдельной записи модели (где **2** — идентификатор для определенной записи). Чтобы выполнить работу, необходимую для отображения записи, нужно создать URL-карту. Функция `reverse()` может преобразовать ваш URL-адрес (в приведенном примере с именем `model-detail-view`) в URL-адрес правильного формата. Конечно, для этого все равно придется прописать соответствие URL-адреса с представлением (`view`) и шаблоном.

### 9.4.4. Методы работы с данными в моделях Django

Когда классы моделей данных определены, с их помощью можно создавать, обновлять или удалять записи, выполнять запросы, а также получать все записи или отдельные подмножества записей из таблиц БД.

Чтобы создать запись в таблице БД, нужно определить (создать) экземпляр модели, а затем вызвать метод `save()`:

```
# Создать новую запись с помощью конструктора модели
a_record = MyModelName(my_field_name="Книга о вкусной еде")
# Сохраните запись в базе данных.
a_record.save()
```

Если поле `MyModelName` предназначено для хранения названий книг, то в соответствующей таблице БД появится запись с названием этой книги: Книга о вкусной еде.

Можно получать доступ к полям в записи БД и изменять их значения. После внесения изменений данных необходимо вызвать метод `save()`, чтобы сохранить измененные значения в базе данных:

```
# Доступ к значениям полей модели
id_book = a_record.id # получить идентификатор записи
name_book = a_record.my_field_name # получить название книги

# Изменить название книги
a_record.my_field_name="Книга о вкусной и здоровой пище"
a_record.save()
```

Вы можете искать в БД записи, которые соответствуют определенным критериям, используя атрибуты объектов модели. Для поиска и выборки данных из БД в Django служит объект `QuerySet`. Это интегрирующий объект — он содержит несколько объектов, которые можно перебирать (прокручивать). Чтобы извлечь все записи из таблицы базы данных, вызовите метод `objects.all()`. Например, следующий код даст возможность получить названия всех книг из БД:

```
all_books = Book.objects.all()
```

Метод `filter()` позволяет отфильтровать данные для `QuerySet` в соответствии с указанным полем и критерием фильтрации. Например, чтобы отфильтровать книги, содержащие в названии слово «дикие» (дикие животные, дикие растения и т. п.), а затем подсчитать их, мы могли бы воспользоваться следующим кодом:

```
wild_books = Book.objects.filter(title__contains='дикие')
n_w_books = Book.objects.filter(title__contains='дикие').count()
```

В этих фильтрах принят формат: `field_name__match_type`, где `field_name` — имя поля, по которому фильтруются данные, а `match_type` — тип соответствия поля определенному критерию ( обратите внимание, что между ними стоит двойное подчеркивание). В приведенном коде полем, по которому фильтруются данные, является название книги (`title`) и задан тип соответствия `contains` (с учетом регистра).

В Django в фильтрах возможны различные типы соответствия (совпадения). Основные типы соответствия приведены в табл. 9.1.

**Таблица 9.1.** Основные типы соответствия (совпадения), используемые в Django

№ п/п	Вызов типа соответствия	Назначение типа соответствия	Аналог на языке SQL
1	<code>exact</code>	Точное совпадение с учетом регистра	<code>SELECT ... WHERE id = 14</code>
2	<code>iexact</code>	Точное совпадение без учета регистра	<code>SELECT ... WHERE name ILIKE 'beatles blog'</code>

Таблица 9.1 (окончание)

№ п/п	Вызов типа соответствия	Назначение типа соответствия	Аналог на языке SQL
3	contains	С учетом регистра	SELECT ... WHERE headline LIKE '%Lennon%'
4	icontains	Без учета регистра	SELECT ... WHERE headline ILIKE '%Lennon%'
5	in	Равно одному из элементов списка или кортежа	SELECT ... WHERE id IN (1, 3, 4)
6	gt	Больше чем	SELECT ... WHERE id > 4
7	gte	Больше или равно	SELECT ... WHERE id >= 4
8	lt	Меньше чем	SELECT ... WHERE id < 4
9	lte	Меньше или равно	SELECT ... WHERE id <= 4
10	range	Внутри диапазона	SELECT ... WHERE id > 4 AND id < 8

Полный список типов соответствий, которые используются в фильтрах, можно посмотреть в оригинальной документации на Django.

Иногда возникает необходимость фильтровать поле, которое определяет отношение «один ко многим» к другой модели. В этом случае вы можете «индексировать» поля в связанной модели с дополнительными двойными подчеркиваниями. Так, например, чтобы выбрать по фильтру книги с определенным жанром, нужно указать имя в поле жанра:

```
books_containing_genre = Book.objects.filter(genre__name__icontains='Фантастика')
```

При выполнении этого кода из БД будут выбраны все книги из жанров: фантастика, научная фантастика.

## 9.5. Формирование моделей данных для сайта «Мир книг»

Итак, мы подошли к тому моменту, когда можно приступить к формированию моделей для сайта «Мир книг». Фактически мы проектируем структуру базы данных: определяем перечень таблиц, создаем в них поля, формируем связи между таблицами. Для этого активируем PyCharm, загрузим ранее созданный проект `World_books` и откроем файл `models.py`, который находится в приложении `catalog`. Путь к этому файлу:

`World_books\WebBooks\catalog\models.py`.

В шаблоне этого программного модуля в верхней части текстового редактора присутствует строка, которая импортирует модуль Django для проектирования моделей (рис. 9.20).

Таким образом, в этом файле уже импортирован базовый класс `models.Model`, от которого будут наследоваться все наши модели данных. А также имеется закомментированная строка, которую можно перевести как «Создавайте свои модели здесь».

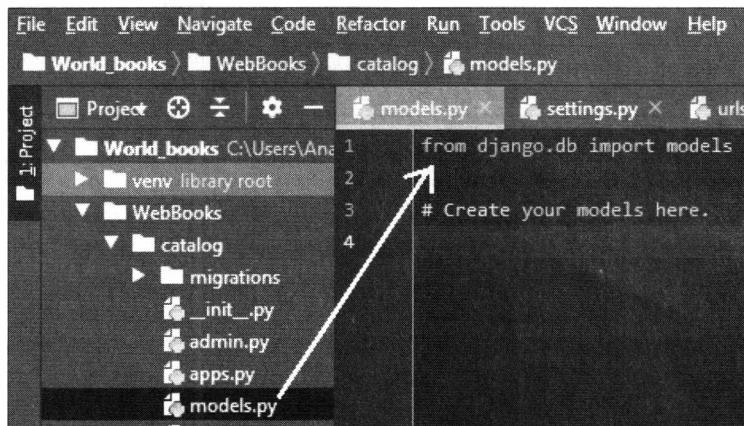


Рис. 9.20. Файл-шаблон models.py для проектирования моделей данных сайта «Мир книг»

Добавим в файл models.py строку импорта класса reverse, который обеспечит получение абсолютных URL-адресов (листинг 9.8, строка выделена серым фоном).

#### Листинг 9.8. Изменения в файле models.py

```

from django.db import models
from django.urls import reverse

```

Эта возможность понадобится для получения ссылки на страницы с деталями списков из самих списков данных. А теперь приступим к созданию моделей для нашего сайта.

### 9.5.1. Модель для хранения жанров книг

Начнем с формирования справочника жанров книг. Для хранения информации о жанрах книг спроектируем модель (таблицу) с именем Genre (листинг 9.9). Эта модель будет содержать всего одно текстовое поле (типа CharField) — name, в котором мы будем хранить наименования жанров книг.

#### Листинг 9.9. Создание модели Genre (жанры книг) в модуле models.py

```

# жанры книг
class Genre(models.Model):
    name = models.CharField(max_length=200,
                           help_text="Введите жанр книги",
                           verbose_name="Жанр книги")
    def __str__(self):
        return self.name

```

Поле name будет использоваться для описания жанра книги, оно ограничено 200 символами и имеет текст подсказки: help\_text. В качестве удобного читаемого имени (verbose\_name) указано значение "Жанр книги", поэтому именно оно будет выводиться на экран в формах рядом с полем name. В конце модели объявлен метод \_\_str\_\_(), возвращающий имя жанра, которое будет внесено в конкретную запись.

## 9.5.2. Модель для хранения языков книг

Продолжим формирование справочников и создадим справочник языков, на которых написаны книги. Для хранения информации о языках спроектируем модель (таблицу) с именем `Language` (листинг 9.10). Эта модель будет содержать всего одно текстовое поле (типа `CharField`) — `name`, в котором мы будем хранить наименования языков.

### Листинг 9.10. Создание модели `Language` (язык книг) в модуле `models.py`

```
# язык книги
class Language(models.Model):
    name = models.CharField(max_length=20,
                           help_text="Введите язык книги",
                           verbose_name="Язык книги")
    def __str__(self):
        return self.name
```

Поле `name` будет использоваться для хранения языка книги, оно ограничено 20 символами и имеет текст подсказки: `help_text`. В качестве удобно читаемого имени (`verbose_name`) указано значение "Язык книги", поэтому именно оно будет выводиться на экран в формах рядом с полем `name`. В конце модели объявлен метод `__str__()`, возвращающий наименование языка книги, которое будет внесено в конкретную запись.

## 9.5.3. Модель для хранения наименования издательства

Продолжим формирование справочников и создадим справочник издательств, которые занимаются публикацией книг. Для хранения информации об издательствах спроектируем модель (таблицу) с именем `Publisher` (листинг 9.11).

### Листинг 9.11. Создание модели `Publisher` (издательство) в модуле `models.py`

```
# издательство
class Publisher(models.Model):
    name = models.CharField(max_length=20,
                           help_text="Введите наименование издательства",
                           verbose_name="Издательство")
    def __str__(self):
        return self.name
```

Как можно видеть в данном листинге, модель `Publisher` содержит одно текстовое поле (типа `CharField`), которое называется `name`. Это поле будет использоваться для хранения наименования издательства, оно ограничено 20 символами и имеет текст подсказки: `help_text`. В качестве удобно читаемого имени (`verbose_name`) указано значение "Издательство", поэтому именно оно будет выводиться на экран в формах рядом с полем `name`. В конце модели объявлен метод `__str__()`, возвращающий наименование издательства, которое будет внесено в конкретную запись.

## 9.5.4. Модель для хранения авторов книг

Теперь создадим справочник, в котором будем хранить сведения об авторах книг. Для хранения информации об авторах спроектируем модель (таблицу) с именем `Author`. Эта модель будет содержать несколько полей: `first_name` — имя автора, `last_name` — фамилия автора, `date_of_birth` — дата его рождения, `about` — информация об авторе, `photo` — фотография (портрет) автора (листинг 9.12).

**Листинг 9.12. Создание модели `Author` (авторы) в модуле `models.py`**

```
# авторы
class Author(models.Model):
    first_name = models.CharField(max_length=100,
                                  help_text="Введите имя автора",
                                  verbose_name="Имя автора")
    last_name = models.CharField(max_length=100,
                                 help_text="Введите фамилию автора",
                                 verbose_name="Фамилия автора")
    date_of_birth = models.DateField(
        help_text="Введите дату рождения",
        verbose_name="Дата рождения",
        null=True, blank=True)
    about = models.TextField(help_text="Введите сведения об авторе",
                            verbose_name="Сведения об авторе")
    photo = models.ImageField(upload_to='images',
                              help_text="Введите фото автора",
                              verbose_name="Фото автора",
                              null=True, blank=True)

    def __str__(self):
        return self.last_name
```

Как можно видеть из данного листинга, модель `Author` содержит текстовые поля (типа `CharField`) для хранения имен и фамилий авторов. Эти поля ограничены 100 символами, имеют текст подсказки: `help_text` и удобно читаемые имена (`verbose_name`). Поле для ввода даты (`date_of_birth` типа `DateField`) предназначено для хранения даты рождения автора книги. Это поле может содержать пустое значение (`null=True`), т. е. не обязательно для заполнения. В модели имеется поле для ввода сведений об авторе (`about` типа `TextField`) и для фотографии или портрета автора (`photo` типа `ImageField`). Поле `photo` имеет атрибут `upload_to='images'`. Это означает, что все файлы с фотографиями или портретами авторов будут автоматически сохраняться в папку проекта с именем `images`. Метод `__str__()` будет возвращать фамилию автора.

## 9.5.5. Модель для хранения книг

Прежде чем проектировать модель для хранения данных о книгах, рассмотрим, какие связи нужно предусмотреть для тех справочников, которые мы уже создали: жанр, язык, издательство и авторы.

Здесь мы исходим из того, что каждая книга будет относиться только к одному жанру. При этом один жанр может соответствовать множеству книг. Значит, здесь присутствует связь «один ко многим» (рис. 9.21).

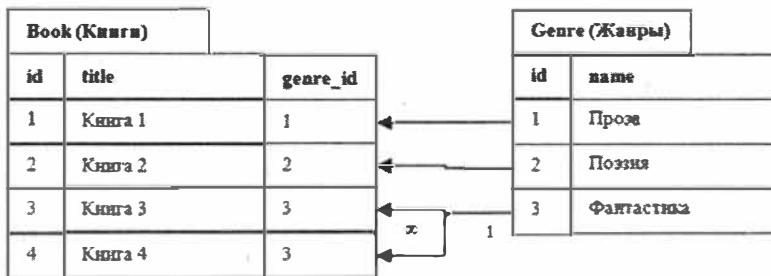


Рис. 9.21. Связь «один ко многим» между жанрами книг и книгами

Записи в таблице `Genre` не повторяются — они уникальны и существуют в единственном экземпляре (в начале линии, показывающей связь, стоит **1**). А вот в таблице `Book` может быть множество книг, которые относятся к одному и тому же жанру (на конце линии связи стоит знак бесконечности). В таком случае `Genre` является *главной* (или родительской) таблицей, а `Book` — *связанной* (или дочерней) таблицей.

Что касается языка, на котором написана книга, то мы исходим из того, что каждая книга написана на одном языке. При этом один язык может соответствовать множеству книг. Значит, здесь тоже присутствует связь «один ко многим» (рис. 9.22).

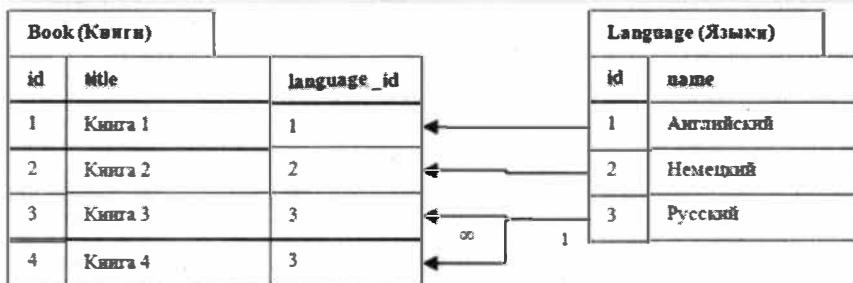


Рис. 9.22. Связь «один-ко-многим» между языками и книгами

Записи в таблице `Language` не повторяются — они уникальны и существуют в единственном экземпляре (в начале линии, показывающей связь, стоит **1**). А вот в таблице `Book` может быть множество книг, которые написаны на одном языке (на конце линии связи стоит знак бесконечности). В таком случае `Language` является *главной* (или родительской) таблицей, а `Book` — *связанной* (или дочерней) таблицей.

Если рассматривать издательство, которое напечатало книгу, то мы будем исходить из того, что каждая книга выпущена одним издательством. При этом одно издательство может соответствовать множеству книг. Значит, здесь тоже присутствует связь «один ко многим» (рис. 9.23).

Записи в таблице `Publisher` не повторяются — они уникальны и существуют в единственном числе (в начале линии, показывающей связь, стоит **1**). А вот в таблице `Book` может быть множество книг, которые изданы одним издательством (на конце линии связи стоит знак бесконечности). В таком случае `Publisher` является *главной* (или родительской) таблицей, а `Book` — *связанной* (или дочерней) таблицей.

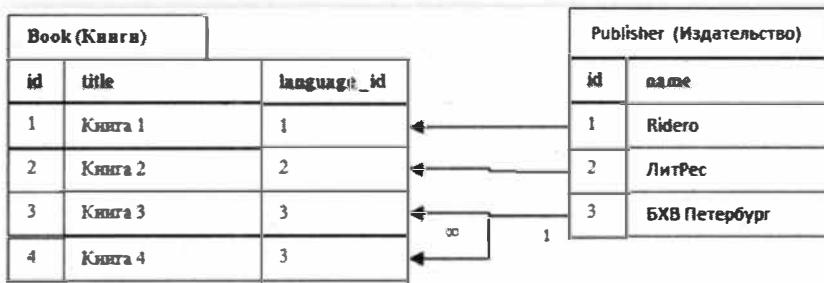


Рис. 9.23. Связь «один ко многим» между издательствами и книгами

Несколько сложнее обстоит дело с книгами и их авторами. Каждая книга может быть написана несколькими авторами, но и автор может написать несколько книг (как самостоятельно, так и в соавторстве). Значит, здесь присутствует связь «многие ко многим», а для реализации такой связи необходим новый объект в модели, или новая связующая таблица в базе данных (рис. 9.24).

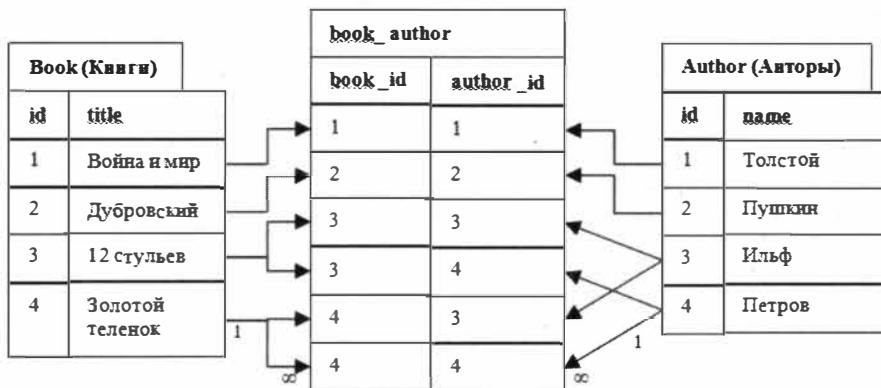


Рис. 9.24. Связь «многие ко многим» между авторами и книгами

Как можно видеть из данного рисунка, книги «Война и мир» и «Дубровский» имеют по одному автору (Толстой и Пушкин). А вот следующие две книги «12 стульев» и «Золотой теленок» — имеют двух авторов (Ильф и Петров). Записи в таблицах Book и Author не повторяются — они уникальны и существуют в единственном числе (в начале линии, показывающей связь, стоит 1). А вот в связывающей их таблице book\_author может быть как множество книг, так и множество авторов (на конце линии связи стоит знак бесконечности). В таком случае таблицы Book и Author являются главными и равнозначными, а book\_author — связывающей таблицей, которая содержит только индексы связанных записей из главных таблиц.

Вот теперь, с учетом рассмотренных связей, можно приступить к созданию таблицы для самих книг. В ней мы будем хранить всю информацию о книгах (название книги, жанр, язык, автор и т. п.), но не о конкретных (физических) экземплярах книг. Каждый экземпляр книги может находиться в различных состояниях, и для этого будет создана отдельная модель (таблица в БД).

Информация о книгах будет содержаться в следующих полях:

- title — название книги;
- genre — жанр книги;
- language — язык;
- publisher — издательство;
- author — автор;
- summary — аннотация книги;
- isbn — уникальный международный номер книги;
- price — цена книги;
- photo — изображение обложки книги.

Описание модели данных о книгах приведено в листинге 9.13.

#### Листинг 9.13. Создание модели Book (книги) в модуле models.py

```
# книги
class Book(models.Model):
    title = models.CharField(max_length=200,
                           help_text="Введите название книги",
                           verbose_name="Название книги")
    genre = models.ForeignKey('Genre',
                           on_delete=models.CASCADE,
                           help_text=" Выберите жанр для книги",
                           verbose_name="Жанр книги", null=True)
    language = models.ForeignKey('Language',
                           on_delete=models.CASCADE,
                           help_text=" Выберите язык книги",
                           verbose_name="Язык книги", null=True)
    publisher = models.ForeignKey('Publisher',
                           on_delete=models.CASCADE,
                           help_text=" Выберите издательство",
                           verbose_name="Издательство", null=True)
    year = models.CharField(max_length=4,
                           help_text="Введите год издания",
                           verbose_name="Год издания")
    author = models.ManyToManyField('Author',
                           help_text="Выберите автора (авторов) книги",
                           verbose_name="Автор (авторы) книги")
    summary = models.TextField(max_length=1000,
                           help_text="Введите краткое описание книги",
                           verbose_name="Аннотация книги")
    isbn = models.CharField(max_length=13,
                           help_text="Должно содержать 13 символов",
                           verbose_name="ISBN книги")
    price = models.DecimalField(decimal_places=2, max_digits=7,
                           help_text="Введите цену книги",
                           verbose_name="Цена (руб.)")
```

```
photo = models.ImageField(upload_to='images',
                         help_text="Введите изображение обложки",
                         verbose_name="Изображение обложки")

def __str__(self):
    return self.title

def get_absolute_url(self):
    # Возвращает URL-адрес для доступа к
    # определенному экземпляру книги
    return reverse('book-detail', args=[str(self.id)])
```

В приведенной модели все поля имеют поясняющий текст (`help_text`) и удобно читаемое имя (`verbose_name`).

Для названия книги создано поле `title` — это текстовое поле типа `CharField`. На название книги наложено ограничение 200 символов.

Для хранения информации о жанре книги создано поле `genre`, которое по первичному ключу связано с моделью `Genre`. Это поле допускает наличие пустого значения.

Для хранения информации о языке, на котором написана книга, создано поле `language`, которое по первичному ключу связано с моделью `Language`. Это поле допускает наличие пустого значения.

Для хранения информации об издательстве, которое выпустило книгу, создано поле `publisher`, которое по первичному ключу связано с моделью `Publisher`. Это поле допускает наличие пустого значения.

Для хранения информации о году издания книги создано поле `year`, это текстовое поле типа `CharField`. На год издания книги наложено ограничение — 200 символов. Это поле допускает наличие пустого значения.

Для хранения информации об авторе (авторах) книги создано поле `author`, которое по первичному ключу связано с моделью `Author`.

Для хранения аннотации книги создано поле `summary` — это текстовое поле типа `CharField`. На аннотацию книги наложено ограничение — 1000 символов.

Для хранения уникального международного номера книги создано поле `isbn` — это текстовое поле типа `CharField`. На уникальный номер книги наложено ограничение — 13 символов.

Для стоимости книги создано поле `price` — это числовое поле типа `DecimalField`. На стоимость книги наложено ограничение — два знака после запятой.

В модели имеется поле `photo` для ввода изображения обложки книги (типа `ImageField`). Поле `photo` имеет атрибут `upload_to='images'`. Это означает, что все файлы с изображениями обложек книг будут автоматически сохраняться в папку проекта с именем `images`.

В данной модели определено несколько методов.

Метод `__str__()` вернет наименование книги.

Метод `get_absolute_url()` вернет URL-адрес, который можно использовать для доступа к детальным записям для этой модели. Например, на странице пользователю выдана строка с названием книги, которое подсвечивается как ссылка на другую страницу.

Если щелкнуть мышью на названии книги, то будет загружена страница, например, с детальной информацией об этой книге. Наличие такой функции позволит сопоставить URL-адреса на страницы, в которых содержится подробная информация о книге, с соответствующим представлением и шаблоном HTML-страницы.

## 9.5.6. Модель для хранения отдельных экземпляров книг и их статуса

Вполне возможно, что одна и та же книга будет представлена несколькими экземплярами, и каждый экземпляр может иметь определенное состояние. Например, на книгу в книжном интернет-магазине сделан заказ, она находится на складе, но еще не доставлена покупателю (стоит в резерве). При этом другие книги тоже лежат на складе, но еще не нашли своего покупателя (выставлены на продажу). Если это библиотека, то конкретный экземпляр книги может находиться в хранилище или в читальном зале, а может быть выдан читателю на дом. Если библиотечная книга выдана читателю, то резонно хранить дату ее возврата. Таким образом, необходимо создать еще одну модель (таблицу в БД), которая будет предназначена для хранения сведений о каждом конкретном экземпляре книги. Назовем эту модель BookInstance (экземпляр книги).

Модель BookInstance будет иметь связи сразу с двумя таблицами: модель данных (таблица) Book, где хранятся сведения о книгах, и модель данных (таблица) Status, где содержится информация о статусе (состоянии) конкретного экземпляра книги (рис. 9.25).

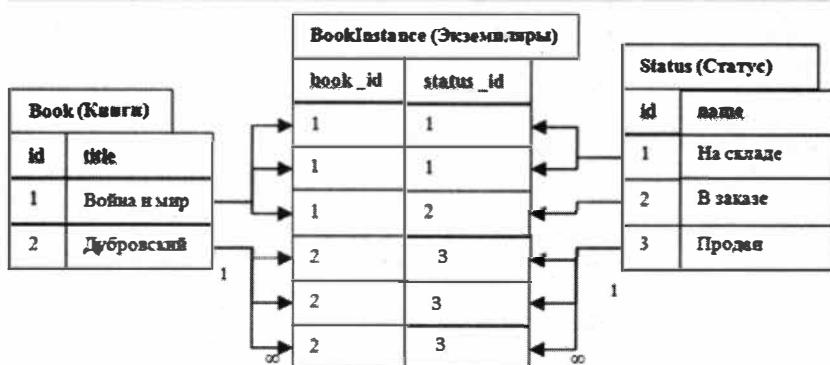


Рис. 9.25. Связи между книгами, экземплярами книг и статусом каждого экземпляра

Как видно из рис. 9.25, в базе данных интернет-магазина имеется информация о трех экземплярах книги «Война и мир» и трех экземплярах книги «Дубровский». При этом два экземпляра книги «Война и мир» находятся на складе, а один экземпляр заказан покупателем, но еще не продан. Все три экземпляра книги «Дубровский» проданы.

Таблицы Book и Status содержат всего по одной записи об объекте, а в таблице BookInstance имеется как множество записей о самих объектах (книгах), так и множество записей об их состоянии (статусе). Таким образом, мы имеем дело со связями «один ко многим». Модель для книг Book у нас уже есть, значит, нам нужно создать еще две модели: Status и BookInstance. Начнем с модели Status (листинг 9.14).

### Листинг 9.14. Создание модели Status (статус) в модуле models.py

```
# состояние экземпляра книги
class Status(models.Model):
    name = models.CharField(max_length=20,
                           help_text="Введите статус экземпляра книги",
                           verbose_name="Статус экземпляра книги")
    def __str__(self):
        return self.name
```

Как можно видеть из листинга 9.14, модель Status содержит одно текстовое поле (типа CharField), которое называется name. Это поле будет использоваться для описания статуса книги — оно ограничено 20 символами и имеет текст подсказки: help\_text. В качестве удобно читаемого имени (verbose\_name) указано значение "Статус экземпляра книги", поэтому именно оно будет выводиться на экран в формах рядом с полем name. В конце модели объявлен метод \_\_str\_\_(), возвращающий значение статуса экземпляра книги, которое будет внесено в конкретную запись.

Теперь можно перейти к созданию модели BookInstance (листинг 9.15).

### Листинг 9.15. Создание модели BookInstance (экземпляр книги) в модуле models.py

```
# экземпляр книги
class BookInstance(models.Model):
    book = models.ForeignKey('Book',
                           on_delete=models.CASCADE, null=True)
    inv_nom = models.CharField(
        max_length=20,
        null=True,
        help_text="Введите инвентарный номер экземпляра",
        verbose_name="Инвентарный номер")
    status = models.ForeignKey('Status',
                           on_delete=models.CASCADE,
                           null=True,
                           help_text='Изменить состояние экземпляра',
                           verbose_name="Статус экземпляра книги")
    due_back = models.DateField(null=True,
                               blank=True,
                               help_text="Введите конец срока статуса",
                               verbose_name="Дата окончания статуса")

    # Метаданные
    class Meta:
        ordering = ["due_back"]
    def __str__(self):
        return '%s %s %s' % (self.inv_nom, self.book, self.status)
```

Для названия книги создано поле book, значение для него будет подгружено из модели Book, с которой имеется связь по первичному ключу.

Для задания статуса экземпляра книги (на складе, в заказе, продана и пр.) предусмотрено поле status, значение для него будет подгружено из модели Status, с которой имеется связь по первичному ключу.

Для задания даты окончания действия статуса для экземпляра книги (например, даты окончания действия заказа) создано поле `due_back` — это поле для ввода дат типа `DateField`.

Для сортировки экземпляров книг создан класс `Meta`, при этом все экземпляры будут отсортированы по дате окончания действия статуса.

Метод `__str__()` представляет объект `BookInstance`, который будет выводить содержимое нескольких полей: название книги, ее инвентарный номер и статус.

Будем считать, что весь приведенный здесь код внесен в файл `models.py`, расположенный по пути: `World_books\WebBooks\catalog\models.py`.

Таким образом, на этом этапе все модели созданы, и можно приступить к миграции данных. Как было отмечено в предыдущей главе, миграция выполняется в два этапа. На первом этапе с помощью менеджера Django формируется программный код, который будет создавать таблицы в БД. Для этого в окне терминала PyCharm выполните команду

```
python manage.py makemigrations
```

В результате будет создан файл миграции: `World_books\WebBooks\catalog\migrations\0001_initial.py` и получено следующее сообщение (рис. 9.26).

```
Terminal: Local +  
Migrations for 'catalog':  
  catalog\migrations\0001_initial.py  
    - Create model Author  
    - Create model Book  
    - Create model Genre  
    - Create model Language  
    - Create model Publisher  
    - Create model Status  
    - Create model BookInstance  
    - Add field genre to book  
    - Add field language to book  
    - Add field publisher to book
```

Рис. 9.26. Создание миграции для сайта «Мир книг»

В этом сообщении говорится, что созданы 7 моделей, добавлены 3 поля.

В файле миграции `0001_initial.py` содержится программный код, обеспечивающий создание таблиц в БД. В этом коде содержатся наименования таблиц, наименования полей и их типы, допускаемые значения, метки полей и т. п. Кроме того, описаны дополнительные ключевые поля, не упоминавшиеся в моделях, но необходимые для корректной работы системы.

Теперь в окне терминала PyCharm выполним команду, которая на основе сформированного файла миграции создаст таблицы в БД:

```
python manage.py migrate
```

Если все было сделано правильно, то в окне терминала PyCharm мы получим сообщение об успешном завершении всех операций, связанных с созданием таблиц в БД (рис. 9.27).

```
Terminal: Local × +
```

```
Apply all migrations: admin, auth, catalog, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying catalog.0001_initial... OK
  Applying sessions.0001_initial... OK
```

Рис. 9.27. Создание таблиц в БД на основе миграции для сайта «Мир книг»

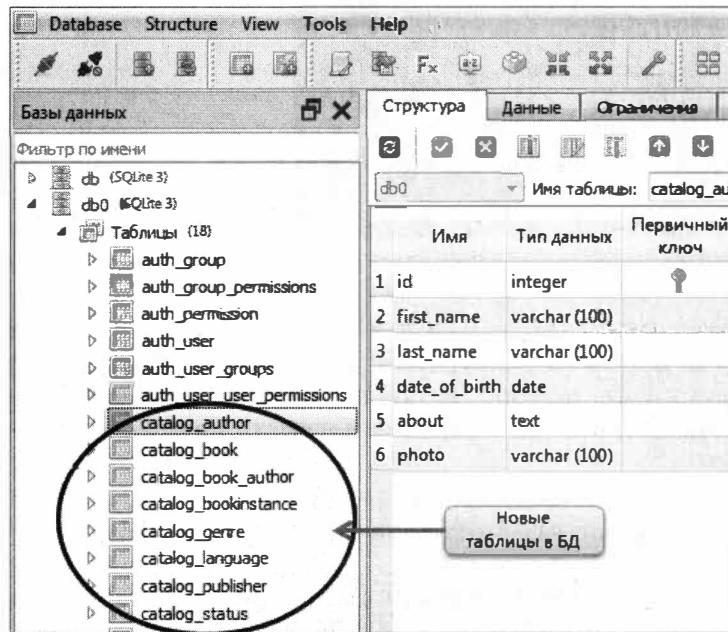


Рис. 9.28. Таблицы в БД, созданные на основе миграции для сайта «Мир книг»

Если теперь с помощью менеджера SQLiteStudio открыть базу данных, то мы увидим, что Django действительно создал все таблицы, которые были спроектированы в моделях данных (рис. 9.28).

Теоретически после этого можно переходить к созданию форм и HTML-страниц нашего сайта, однако перед этим мы познакомимся с еще одним важным элементом Django — встроенной административной панелью.

## 9.6. Административная панель Django Admin

Теперь, когда модели для сайта «Мир книг» созданы, добавим некоторые «настоящие» данные о книгах, воспользовавшись для этого административной панелью Django Admin. По ходу дела мы разберемся с тем, как зарегистрировать в ней модели, как войти и внести данные, а также рассмотрим некоторые способы дальнейшего улучшения вида административной панели.

Приложение Django Admin позволяет на основе ранее разработанных моделей данных автоматически сформировать некоторую часть сайта, предназначенную для создания, просмотра, обновления и удаления записей. Это может сэкономить достаточно много времени в процессе разработки сайта, упрощая тестирование ваших моделей на предмет правильности данных. Это также может быть полезным для управления данными на стадии публикации сайта. Разработчики Django рекомендуют использовать это приложение только для управления данными на уровне разработчиков или владельцев сайтов (администраторов, специалистов внутри организации, владеющей сайтом). Это связано с тем, что модельно-ориентированный подход не является лучшим интерфейсом для внешних пользователей, которые, кроме всего прочего, неумелыми действиями могут нарушить структуру моделей данных.

Все необходимые настройки, которые необходимо включить в административное приложение веб-сайта, были сделаны автоматически, когда создавался веб-проект. Остается только добавить в приложение Django Admin разработанные модели данных, т. е. просто зарегистрировать их. После регистрации моделей нужно создать «суперпользователя» (главного администратора сайта), и уже он сможет войти на сайт и вносить в БД записи тестовых данных (книги, авторы, экземпляры книг, жанры и т. д.). Это будет полезным для тестирования представлений и шаблонов, создаваемых при разработке интерфейса сайта для внешних пользователей.

### 9.6.1. Регистрация моделей данных в Django Admin

Для регистрации моделей в административной части сайта нужно открыть файл `admin.py` по пути: `World_books\WebBooks\catalog\admin.py`.

Изначально в нем ничего нет, кроме одной строки импорта модуля `django.contrib.admin` (рис. 9.29).

Зарегистрируем модели, добавив в файл `admin.py` код листинга 9.16.

#### Листинг 9.16. Измененный код модуля `admin.py`

```
from .models import Author, Book, Genre, Language, Publisher, Status, BookInstance
admin.site.register(Author)
```

```
admin.site.register(Book)
admin.site.register(Genre)
admin.site.register(Language)
admin.site.register(Publisher)
admin.site.register(Status)
admin.site.register(BookInstance)
```

Этот код просто импортирует созданные нами модели, а затем в нескольких строках

```
admin.site.register(<имя модели>)
```

вызывает модуль `admin.site.register(...)` для регистрации каждой из них.

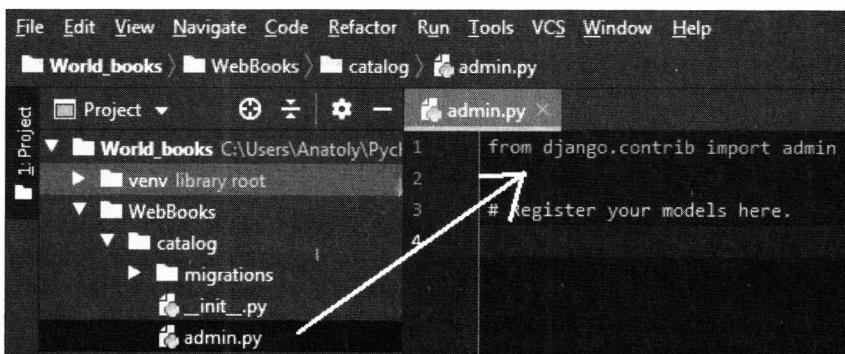


Рис. 9.29. Модуль регистрации моделей `admin.py`

## 9.6.2. Работа с данными в Django Admin

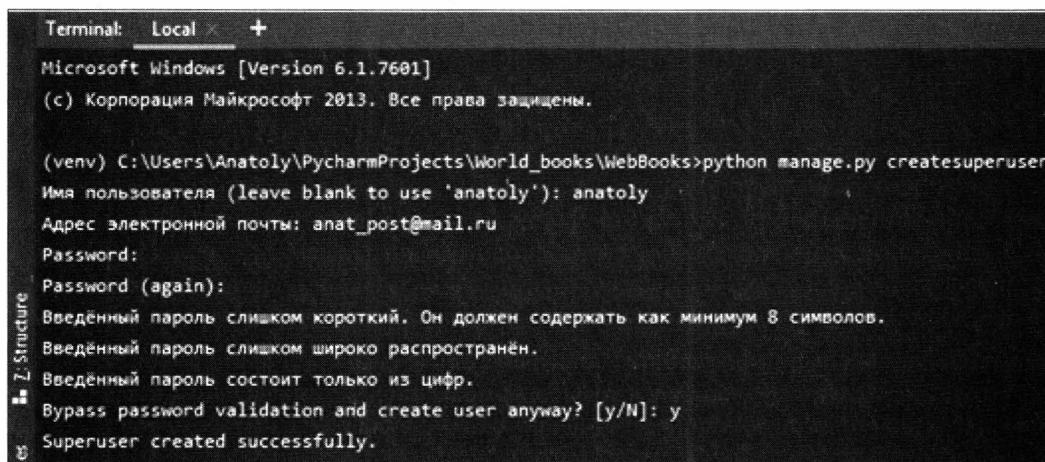
Чтобы войти в административную панель Django, необходимо иметь учетную запись пользователя со статусом Staff (персонал). Для просмотра и создания записей пользователю также понадобится разрешение на управление всеми нашими объектами. Прежде всего, необходимо создать учетную запись superuser (суперпользователь или главный администратор), которая даст полный доступ к сайту и все необходимые разрешения.

Для создания суперпользователя воспользуйтесь модулем `manage.py` и выполните следующую команду в окне терминала PyCharm:

```
python manage.py createsuperuser
```

После ввода этой команды Django автоматически сформирует имя пользователя (по умолчанию это имя компьютера). Можно оставить предложенное имя и нажать клавишу <Enter>, а можно набрать другое имя суперпользователя и подтвердить его также нажатием клавиши <Enter>.

Затем вам будет предложено ввести электронный адрес суперпользователя и дважды ввести пароль для входа суперпользователя в административную панель. Если введенный вами пароль не удовлетворит требованиям системы (слишком простой, слишком короткий и пр.), будет выдано соответствующее предупреждение. После проверки валидности пароля нужно будет нажать клавишу <Y>, после чего вы получите сообщение, что суперпользователь успешно создан (рис. 9.30).



```
Terminal: Local +  
Microsoft Windows [Version 6.1.7601]  
(c) Корпорация Майкрософт 2013. Все права защищены.  
  
(venv) C:\Users\Anatoly\PycharmProjects\World_books\WebBooks>python manage.py createsuperuser  
Имя пользователя (leave blank to use 'anatoly'): anatoly  
Адрес электронной почты: anat_post@mail.ru  
Password:  
Password (again):  
Введённый пароль слишком короткий. Он должен содержать как минимум 8 символов.  
Введённый пароль слишком широко распространён.  
Введённый пароль состоит только из цифр.  
Bypass password validation and create user anyway? [y/N]: y  
Superuser created successfully.
```

Рис. 9.30. Создание суперпользователя сайта «Мир книг» в окне терминала PyCharm

Теперь можно запустить сервер и протестировать вход на административную панель сайта. Для этого выполните в окне терминала PyCharm команду

```
python manage.py runserver
```

Щелкнув мышью в окне терминала PyCharm на ссылку <http://127.0.0.1:8000/> (или набрав этот URL в адресной строке браузера), мы окажемся на главной странице сайта. Для входа в панель администратора сайта нужно в адресной строке набрать ссылку на соответствующую страницу: [/admin](http://127.0.0.1:8000/admin) (в нашем случае: <http://127.0.0.1:8000/admin>). После этого откроется окно для идентификации суперпользователя (рис. 9.31).

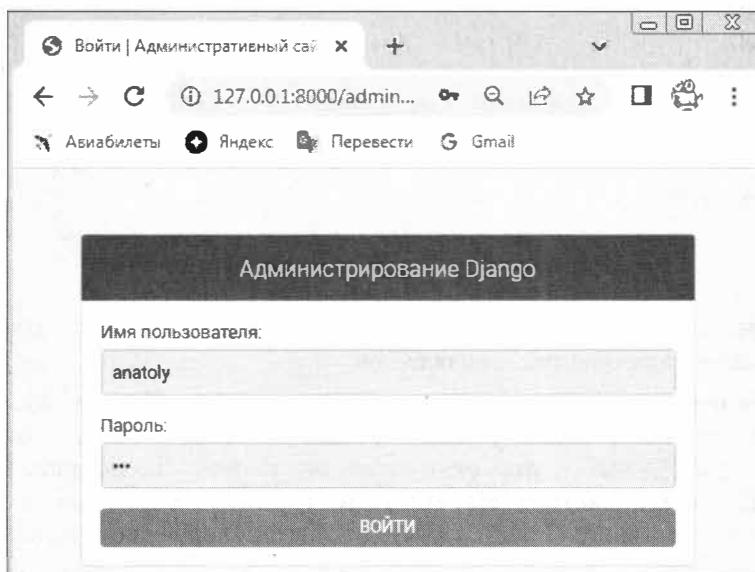


Рис. 9.31. Окно для входа суперпользователя на страницу администратора сайта «Мир книг»

Введите в соответствующие поля этого окна имя суперпользователя и его пароль, и будет открыто главное окно администратора сайта (рис. 9.32).

В окне администрирования сайта отображаются все модели приложения. Здесь можно щелкнуть левой кнопкой мыши на названии модели, чтобы получить список всех связанных записей, а затем щелкнуть по одной из этих записей для ее редактирования. Вы также можете непосредственно перейти по ссылкам **Добавить** или **Изменить**, расположенным рядом с каждой моделью, чтобы начать создание или изменение записи этого типа.

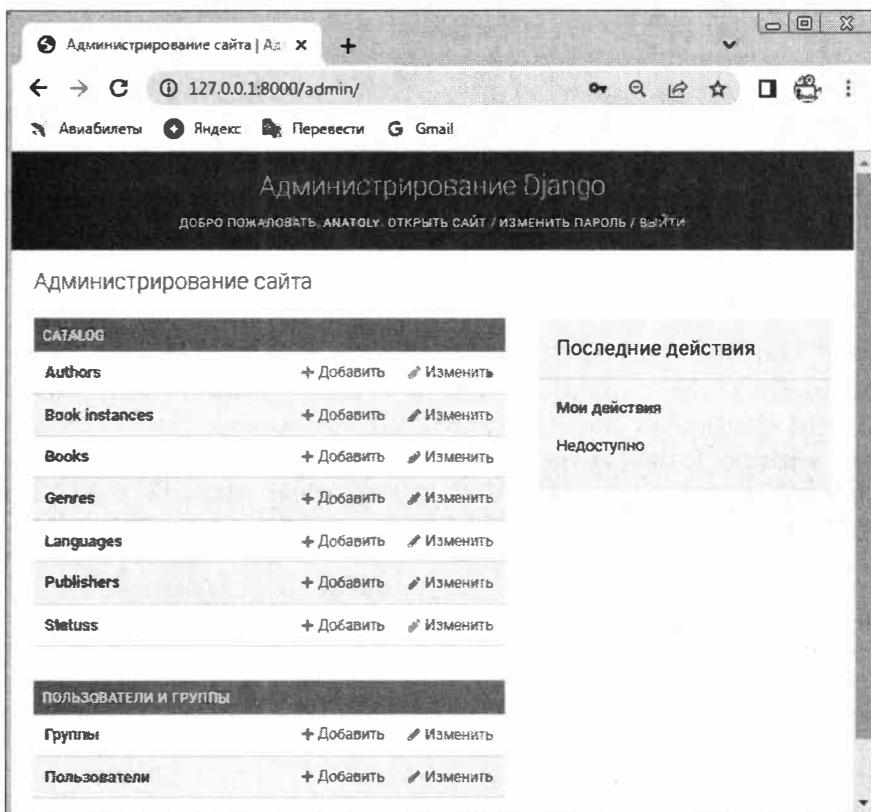


Рис. 9.32. Главное окно администратора сайта «Мир книг»

Итак, мы зарегистрировались, создали учетную запись суперпользователя и вошли с ее помощью в панель администрирования сайта.

Приступим теперь к формированию базовых справочников. Щелкнем мышью на ссылке **Добавить** рядом с моделью **Genres**, описывающей жанры книг, — откроется окно для ввода жанров. Введем в этом окне несколько жанров: **Детективы**, **Поэзия**, **Приключения**, **Проза**, **Фантастика**, **Научно-техническая**, **Учебная**, нажимая после ввода очередного жанра клавишу <Enter>. После ввода всех жанров получим следующее окно (рис. 9.33).

Если теперь вернуться в главное окно администратора сайта, то вам будут показаны последние действия, которые выполнялись с моделями (рис. 9.34).

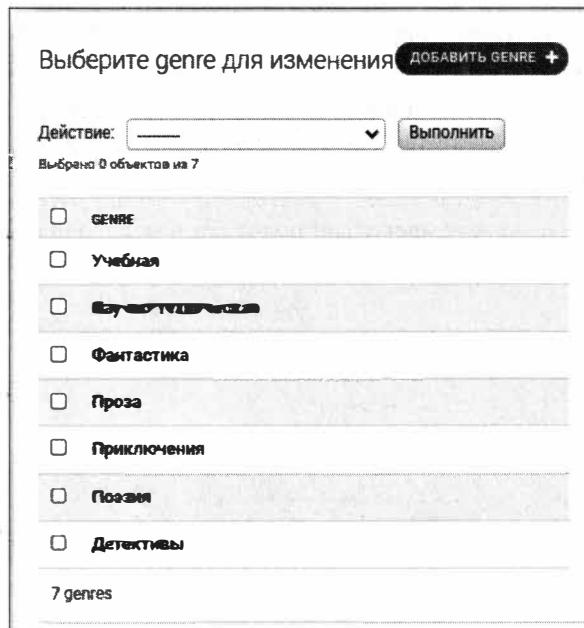


Рис. 9.33. Окно администратора сайта после ввода семи жанров книг

Администрирование Django

ДОБРО ПОЖАЛОВАТЬ, ANATOLY | ОТКРЫТЬ САЙТ / ИЗМЕНИТЬ ПАРОЛЬ / ВЫЙТИ

## Администрирование сайта

CATALOG

Authors	+ Добавить	Изменить
Book instances	+ Добавить	Изменить
Books	+ Добавить	Изменить
Genres	+ Добавить	Изменить
Languages	+ Добавить	Изменить
Publishers	+ Добавить	Изменить
Statuss	+ Добавить	Изменить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

Группы	+ Добавить	Изменить
Пользователи	+ Добавить	Изменить

**Последние действия**

**Мои действия**

- + Учебная Genre
- + Научно-техническая Genre
- + Фантастика Genre
- + Проза Genre
- + Приключения Genre
- + Поззия Genre
- + Детективы Genre

Рис. 9.34. Ведение журнала действий администратора сайта

Теперь аналогичным образом сформируем справочник языков, на которых написаны книги: **Русский, Английский, Немецкий, Французский**.

Добавим книжные издательства: **Ridero, ЛитРес, БХВ-Петербург, Просвещение**.

Введем сведения о некоторых авторах: **Александр Беляев, Александр Пушкин, Лев Толстой, Илья Ильф, Евгений Петров**.

Поскольку модель `Autors` имеет несколько полей, то при вводе сведений об авторах мы получим окно несколько иного вида (рис. 9.35).

Добавить author

Имя автора: Александр  
Введите имя автора

Фамилия автора: Беляев  
Введите фамилию автора

Дата рождения: 16.03.1882 Сегодня |

Введите дату рождения

Внимание: Ваше локальное время отстает от времени сервера на 3 часа.

**Сведения об авторе:**

Александр Романович Беляев (16 марта 1884, Смоленск, Смоленская губерния). Русский и советский писатель-фантаст, репортер и адвокат, журналист. Один из основоположников советской научно-фантастической литературы, первый из советских писателей, целиком посвятивший себя этому жанру.  
Среди наиболее известных его романов: «Голова профессора Доуэля», «Человек-амфибия», «Аризель», «Звезда КЭЦ» и многие другие (всего более 70 научно-фантастических произведений, в том числе 17 романов). За значительный вклад в русскую фантастику и пророческие идеи Беляева называют «русским Жюлем Верном».

Введите сведения об авторе

Фото автора: Выберите файл   
Введите фото автора

Рис. 9.35. Окно администратора сайта для ввода сведений об авторах

В этом окне для ввода дат будет автоматически подключаться календарь. Дату можно либо выбрать из календаря, либо ввести с клавиатуры. Если при ручном вводе даты будет сделана ошибка, то Django выдаст соответствующее предупреждение. Поле для ввода сведений об авторах имеет достаточно большой размер, кроме того, его можно расширить (сузить), зацепив мышью и переместив метку в правом нижнем углу поля.

Сформировав базовые справочники, можно перейти к заполнению данными связанных с ними таблиц. Введем в базу данных несколько книг. Для этого щелкнем левой кноп-

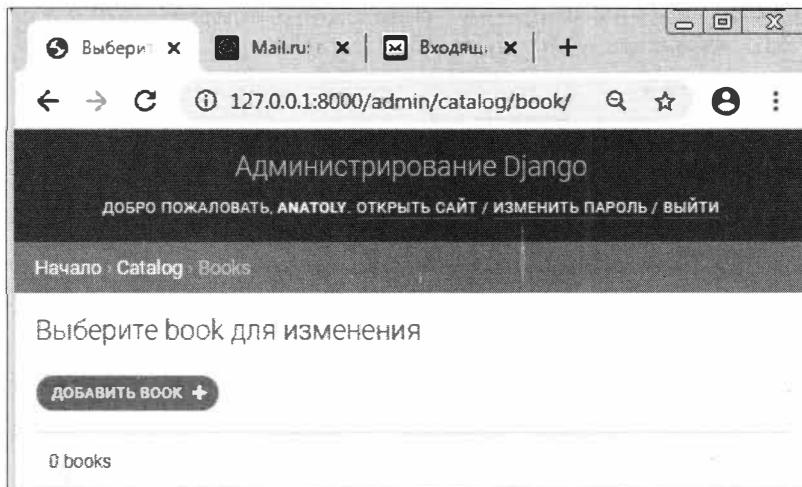


Рис. 9.36. Начальное окно администратора сайта для добавления новой книги

кой мыши на ссылке **Добавить** рядом с моделью **Books** — откроется следующее окно (рис. 9.36).

В этом окне видно, что на текущий момент в БД нет ни одной книги, но при этом существует кнопка **ДОБАВИТЬ BOOK +**, позволяющая войти в режим ввода сведений о книгах. После нажатия на эту кнопку будет открыто основное окно для ввода сведений о книгах (рис. 9.37).

Вспомним, что модель «Книги» связана связью «один ко многим» со следующими моделями:

- «Жанр книги»;
- «Язык книги»;
- «Издательство».

Следовательно, один жанр — много книг, один язык — много книг, одно издательство — много книг. В связи с этим для выбора жанра, языка и издательства Django создает поле выбора в виде выпадающего списка. А вот книги и авторы имеют связь «многие ко многим». Поэтому для выбора автора (авторов) книги предназначено поле в виде простого списка, в котором можно одновременно выбрать несколько авторов. Рядом с этим полем имеется подсказка — для выбора нескольких авторов необходимо щелкнуть в списке на фамилии автора, удерживая при этом нажатой клавишу <Ctrl>.

Рядом с полями ввода жанра книги, языка и автора книги имеется значок +. Если щелкнуть по нему левой кнопкой мыши, то появится новое окно, в котором можно пополнить соответствующий справочник.

Введем в этом окне хотя бы по одной книге каждого автора. А для Ильфа и Петрова введем две книги, чтобы проверить, как будет работать связь между таблицами «многие ко многим». Результаты ввода сведений о книгах показаны на рис. 9.38.

Если щелкнуть мышью на названии книги, то будет показано окно с полной информацией об этой книге (рис. 9.39).

**Добавить book**

**Название книги:**  Введите название книги

**Жанр книги:**  Выберите жанр для книги

**Язык книги:**  Выберите язык книги

**Издательство:**  Выберите издательство

**Год издания:**  Введите год издания

**Автор книги:**  Выберите автора книги Удерживайте "Совет" (или "Command" на Mac), чтобы выбрать несколько элементов.

**Аннотация книги:**  Введите краткое описание книги

**ISBN книги:**  Должно содержать 13 символов

**Цена (руб.):**  Введите цену книги

**Изображение обложки:**  Файл не выбран  
Введите изображение обложки

Рис. 9.37. Основное окно администратора сайта для добавления информации о книгах

**Выберите book для изменения**

**Действие:**  Выполнить

Выбрано 0 объектов из 5

<input type="checkbox"/> <b>book</b>
<input type="checkbox"/> Продавец воздуха
<input type="checkbox"/> Дубровский
<input type="checkbox"/> Война и мир
<input type="checkbox"/> Золотой теленок
<input type="checkbox"/> 12 стульев

5 books

Рис. 9.38. Окно администратора сайта с добавленной информацией о пяти книгах

Изменить book

**Война и мир**

**ИСТОРИЧ** **СМОТРЕТЬ НА САЙТЕ >**

**Название книги:** Война и мир

**Жанр книги:** Проза

**Язык книги:** Русский

**Издательство:** ЛитРес

**Год издания:** 1985

**Автор книги:** Беляев, Пушкин, Толстой, Ильф

**Аннотация книги:**

"Война и мир" Л. Н. Толстого - книга на все времена. Как будто она существовала всегда, настолько знакомым кажется текст, едва мы открываем первые страницы романа, настолько памятны многие его эпизоды: охота и свадьки, первый бал Наташи Ростовой, лунная ночь в Отрадном, князь Андрей в сражении при Аusterлице... Сцены "мирной", семейной жизни

**ISBN книги:** 978-5-389-062

**Цена (руб.):** 2500,00

**Изображение обложки:** На данный момент: Война\_и\_мир.jpg  
Кашель: Выберите файл | Файл не выбран  
Выгрузить изображение обложки

Рис. 9.39. Окно администратора сайта с информацией о книге

Итак, мы успешно завели в БД несколько книг. Теперь добавим несколько записей в таблицу со статусом экземпляров книг. Для этого щелкнем мышью на знаке «плюс» рядом с надписью **Statusss**. После этого откроется окно для ввода статуса экземпляров книг (рис. 9.40).

Введем в данную таблицу три статуса (состояния) экземпляров книг: **На складе**, **В заказе**, **Продана** (рис. 9.41).

Теперь введем сведения о конкретных экземплярах книг и их статусе. Для этого в главном окне администратора сайта щелкнем мышью на ссылке **Добавить** рядом с моделью **Book Instances**, после чего откроется соответствующее окно (рис. 9.42).

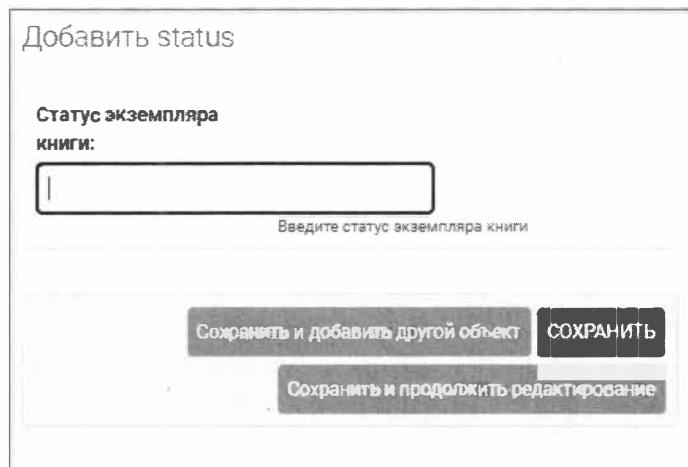


Рис. 9.40. Окно администратора сайта для ввода информации о статусе экземпляров книг

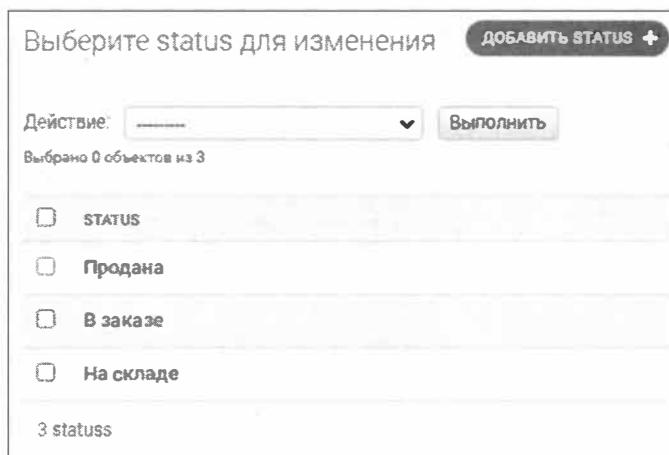


Рис. 9.41. Результаты ввода информации о статусе экземпляров книг

Здесь можно задать параметры для каждого отдельного экземпляра книги (инвентарный номер) и указать статус этого экземпляра. Для тестирования нашего приложения введем информацию о нескольких экземплярах каждой книги. Результаты ввода данных об экземплярах книг представлены на рис. 9.43.

Итак, мы заполнили информацией все наши модели, а вернее, соответствующие таблицы в БД. Мы можем убедиться в том, что информация действительно записана в БД, если откроем содержимое таблиц БД с помощью менеджера SQLiteStudio (рис. 9.44).

Таким образом, используя административную панель, мы смогли заполнить таблицы БД информацией через веб-интерфейс, при этом, не создав ни одной HTML-страницы, и не прибегая к SQL-запросам. Весь необходимый интерфейс был нам предоставлен внутренней структурой Django. И это еще не все — в Django имеется возможность изменить настройки и внешний вид этого интерфейса, а как это делается, описано в следующем разделе.

Добавить book instance

Book:  — + •

Инвентарный номер:  
  
Введите инвентарный номер экземпляра

Статус экземпляра книги:  
 — + •  
Изменить состояние экземпляра

Дата окончания статуса:  
 СегоднЯ CALENDAR  
Введите конец срока статуса

Внимание: Ваше локальное время отстает от времени сервера на 3 часа.

Сохранить и добавить другой объект СОХРАНИТЬ  
Сохранить и продолжить редактирование

Рис. 9.42. Окно администратора сайта для ввода информации об экземплярах книг

Выберите book instance для изменения Добавить BOOK INSTANCE +

Действие:  Выполнить

Выбрано 0 объектов из 9

<input type="checkbox"/>	BOOK INSTANCE
<input type="checkbox"/>	1 Дубровский На складе
<input type="checkbox"/>	2 Война и мир Продана
<input type="checkbox"/>	1 Война и мир На складе
<input type="checkbox"/>	2 Золотой теленок В замазе
<input type="checkbox"/>	1 Золотой теленок На складе
<input type="checkbox"/>	3 12 стульев На складе
<input type="checkbox"/>	2 12 стульев На складе
<input type="checkbox"/>	1 12 стульев На складе
<input type="checkbox"/>	2 Дубровский В заказе

9 book instances

Рис. 9.43. Окно администратора сайта после ввода информации об экземплярах книг

The screenshot shows the MySQL Workbench interface with the 'catalog\_author' table selected. The table has columns: id, first name, last name, date of birth, about, and photo. The data is as follows:

	first name	last name	date of birth	about	photo
1	Александр	Беляев	1882-03-15	Александр Романович Беляев (16 марта 1884, Смоленск, Смоленская губерния). Русский и советский писатель.	Беляев_A.jpg
2	Александр	Пушкин	1799-05-26	Александр Сергеевич Пушкин (26 мая 1799, Москва) — русский поэт, драматург и прозаик, заложивший основы классической русской литературы.	Пушкин_A.jpg
3	Лев	Толстой	1828-08-28	Лев Николаевич Толстой (28 августа 1828, Ясная Поляна, Тульская губерния). Один из наиболее известных русских писателей.	Толстой_Л.jpg
4	Илья	Ильф	1897-10-03	Илья Арнольдович Ильф (при рождении Иехил-Лейб Альберт Файзильберг; 3 октября 1897 года, Одесса).	Ильф_И.jpg
5	Евгений	Петров	1902-10-30	Евгений Петрович Петров (настоящая фамилия — Катаев; 30 ноября 1902). Русский советский писатель, сценарист.	Петров_E.jpg

Рис. 9.44. Содержимое таблиц БД после ввода необходимой информации через панель администрирования сайта

## 9.7. Изменение конфигурации административной панели Django

Административная панель Django обеспечивает эффективную работу с БД разработчикам сайтов, используя информацию из зарегистрированных моделей данных:

- каждая модель имеет набор записей в виде строк, создаваемых методом `_str_()` модели, и связанных с представлением (`view`) для их редактирования. По умолчанию в верхней части этого представления находится меню действий, которое позволяет удалить несколько записей за раз;
- формы для редактирования и добавления записей содержат все поля модели, которые расположены вертикально в порядке их объявления в модели.

Однако для упрощения работы с административной панелью можно настроить интерфейс пользователя. Вот некоторые доступные настройки:

- List views** (список представлений):
  - добавление дополнительных отображаемых полей или информации для каждой записи;
  - добавление фильтров для отбора записей по разным критериям (например, статус выдачи книги);
  - добавление дополнительных вариантов выбора в меню действий и места расположения этого меню на форме.
- Detail views** (подробное представление):
  - выбор отображаемых полей, их порядка, группирования и т. д.;
  - добавление связанных полей к записи (например, возможности добавления и редактирования записей книг при создании записи автора).

В этом разделе как раз будут рассмотрены некоторые желательные изменения для совершенствования интерфейса пользователя нашего сайта «Мир книг». В частности, включение дополнительной информации в списки моделей Book и Author, а также улучшение расположения элементов соответствующих представлений редактирования.

### 9.7.1. Регистрация класса *ModelAdmin*

Для изменения отображения модели в пользовательском интерфейсе административной панели необходимо определить класс *ModelAdmin* (он описывает расположение элементов интерфейса, где *Model* — наименование модели) и зарегистрировать его для использования с этой моделью.

Откройте файл *admin.py* в каталоге приложения (по пути *World\_books\WebBooks\catalog\admin.py*). Код модуля *admin.py* нужно изменить так, как показано в листинге 9.17.

#### Листинг 9.17. Измененный код модуля *admin.py*

```
from django.contrib import admin
from .models import Author, Book, Genre, Language, \
    Publisher, Status, BookInstance

# Определяем класс AuthorAdmin для авторов книг
class AuthorAdmin(admin.ModelAdmin):
    pass
# admin.site.register(Author)
# Регистрируем класс AuthorAdmin для авторов книг
admin.site.register(Author, AuthorAdmin)
# admin.site.register(Book)
# Регистрируем класс BookAdmin для книг
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    pass
# admin.site.register(BookInstance)
# Регистрируем класс BookInstanceAdmin для экземпляров книг
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    pass
admin.site.register(Genre)
admin.site.register(Language)
admin.site.register(Publisher)
admin.site.register(Status)
```

В данный модуль были внесены следующие изменения:

- Закомментирована исходная регистрация модели *Author* с добавлением префикса #:

```
# admin.site.register(Author)
```

- Добавлен новый класс *AuthorAdmin* и выполнена его регистрация:

```
# Определяем класс AuthorAdmin для авторов книг
```

```
class AuthorAdmin(admin.ModelAdmin):
```

```
    pass
```

```
# Регистрируем класс AuthorAdmin для авторов книг
admin.site.register(Author, AuthorAdmin)
```

- Закомментированы классы ModelAdmin для моделей Book и BookInstance:

```
# admin.site.register(Book)
# admin.site.register(BookInstance)
```

- Созданы новые классы Book и BookInstance. В этот раз для создания и регистрации новых моделей мы воспользовались декоратором @register (он делает то же самое, что и метод admin.site.register()):

```
# Регистрируем класс BookAdmin для книг
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    pass

# Регистрируем класс BookInstanceAdmin для экземпляров книг
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    pass
```

Пока что все наши admin-классы пустые (содержат только инструкцию pass), поэтому в интерфейсе панели администратора сайта ничего не изменится. Добавим код для задания особенностей интерфейса моделей.

## 9.7.2. Настройка отображения списков

В текущий момент приложение «Мир книг» отображает всех авторов, используя имя объекта, возвращаемое методом `_str_()` модели. Это приемлемо, когда есть только несколько авторов и у всех разные фамилии. Однако если авторов много, то возможно появление как бы дублирующих записей, например, если есть несколько авторов с фамилией Иванов. Чтобы показать большее число полей, изменить порядок их отображения или ввести дополнительные поля, можно создать новый кортеж `list_display`.

Заменим класс `AuthorAdmin` в модуле `admin.py` кодом, приведенным в листинге 9.18.

### Листинг 9.18. Измененный код модуля `admin.py`

```
# Определяем класс AuthorAdmin для авторов книг
class AuthorAdmin(admin.ModelAdmin):
    list_display = ('last_name', 'first_name', 'date_of_birth', 'photo')
```

Здесь названия полей, которые будут отображаться в списке, приведены в кортеже `list_display` в требуемом порядке (это те же имена полей, что и в исходной модели).

Если теперь перезапустить сайт и перейти к списку авторов, то указанные поля должны отображаться в виде таблицы (рис. 9.45).

Обратите внимание на поле, в котором показана дата рождения. Например, в БД дата рождения Евгения Петрова хранится в формате «1902-10-30», а в таблице отображается с учетом локализации сайта «30 октября 1902 г.». Напомним, что в файле `settings.py` параметр локализации имеет значение `LANGUAGE_CODE = 'ru-ru'`.

Выберите author для изменения				<b>Добавить AUTHOR +</b>
Действие:		<b>Выполнить</b>		Выбрано 0 объектов из 5
<input type="checkbox"/>	ИМЯ АВТОРА	ФАМИЛИЯ АВТОРА	ДАТА РОЖДЕНИЯ	ФОТО АВТОРА
<input type="checkbox"/>	Евгений	Петров	30 октября 1902 г.	Петров_Е.jpg
<input type="checkbox"/>	Илья	Ильф	3 октября 1897 г.	Ильф_И.jpg
<input type="checkbox"/>	Лев	Толстой	28 августа 1828 г.	Толстой_Л.jpg
<input type="checkbox"/>	Александр	Пушкин	26 мая 1799 г.	Пушкин_А.jpg
<input type="checkbox"/>	Александр	Беляев	16 марта 1882 г.	Беляев_А.jpg

5 authors

Рис. 9.45. Сведения об авторах, показанные в виде таблицы с помощью кортежа list\_display

Для модели, описывающей книги (Book), в административной панели отображалось только название книги, добавим отображение еще трех полей: genre, language и author. Поля genre (жанр) и language (язык) имеют внешний ключ (ForeignKey), который создает связь «один ко многим», поэтому в этих полях вместо идентификаторов (id) жанра и языка будут показаны значения \_\_str\_\_() из связанных таблиц (т. е. не идентификаторы, а значения жанра и языка книги).

К сожалению, мы не можем напрямую показать поле author, т. к. между авторами и книгами имеется связь «многие ко многим» (ManyToManyField), поскольку у одной книги может быть несколько авторов. Для того чтобы сформировать список авторов книги, мы добавим функцию (метод) display\_author в модели данных Book. Эта функция станет формировать строку, в которой будут представлены все авторы книги. Откроем файл WebBooks\catalog\models.py и к классу книги (класс Book) добавим функцию (метод) с именем display\_author (листинг 9.19).

#### Листинг 9.19. Измененный код модуля models.py

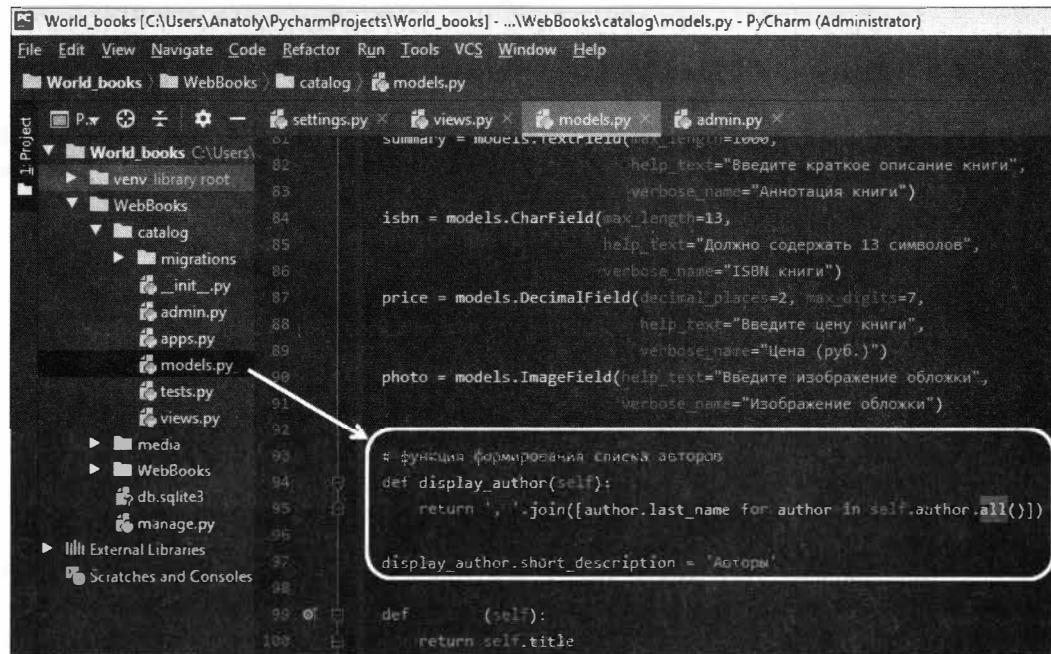
```
# функция формирования списка авторов
def display_author(self):
    return ', '.join([author.last_name for author in
                     self.author.all()])
display_author.short_description = 'Авторы'
```

Эту функцию необходимо вставить в модель Book в файле models.py так, как показано на рис. 9.46.

В этой функции организован цикл с инструкцией for. В теле цикла выбираются все авторы, связанные с данной книгой, из них формируется список, который будет возвращен в точку вызова. Для этого списка формируется короткое имя «Авторы»:

```
display_author.short_description = 'Авторы'
```

Теперь необходимо заменить класс BookAdmin в файле WebBooks\catalog\admin.py на версию, приведенную в листинге 9.20.



```

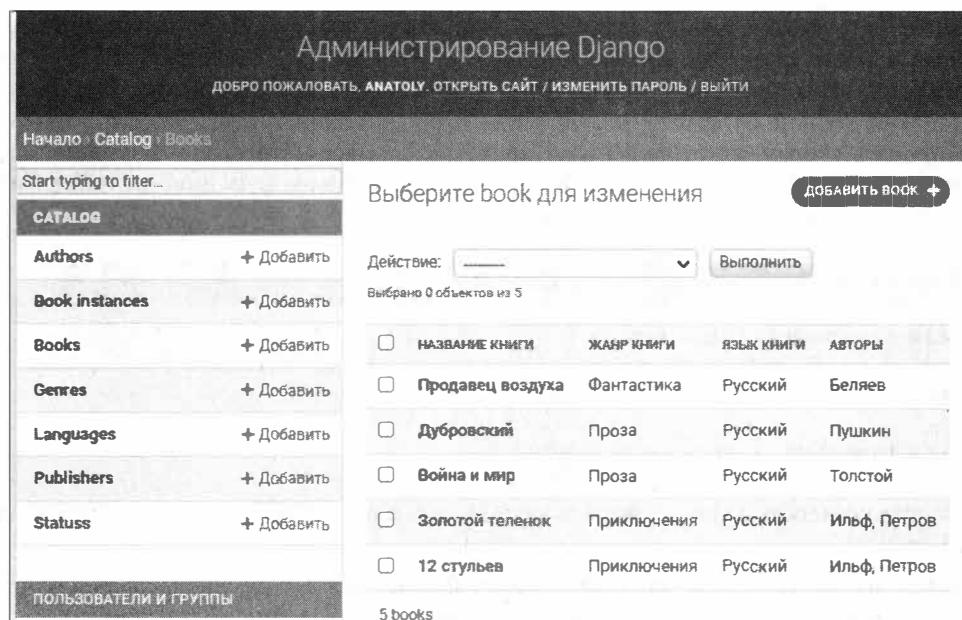
World_books [C:\Users\Anatoly\PycharmProjects\World_books] - ...\\WebBooks\\catalog\\models.py - PyCharm (Administrator)
File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books WebBooks catalog models.py
Project World_books C:\Users\Anatoly\PycharmProjects\World_books
  -> venv library root
    -> WebBooks
      -> catalog
        -> migrations
          -> __init__.py
          -> admin.py
          -> apps.py
          -> models.py <-- This file
          -> tests.py
          -> views.py
        -> media
      -> WebBooks
      -> db.sqlite3
      -> manage.py
  -> External Libraries
  -> Scratches and Consoles
  -> 93
  -> 94
  -> 95
  -> 96
  -> 97
  -> 98
  -> 99
  -> 100
settings.py × views.py × models.py × admin.py ×
summary = models.TextField(max_length=1000,
                           help_text="Введите краткое описание книги",
                           verbose_name="Аннотация книги")
isbn = models.CharField(max_length=13,
                        help_text="Должно содержать 13 символов",
                        verbose_name="ISBN книги")
price = models.DecimalField(decimal_places=2, max_digits=7,
                            help_text="Введите цену книги",
                            verbose_name="Цена (руб.)")
photo = models.ImageField(help_text="Введите изображение обложки",
                           verbose_name="Изображение обложки")

# функция формирования списка авторов
def display_author(self):
    return ', '.join([author.last_name for author in self.author.all()])
display_author.short_description = 'Авторы'

def __str__(self):
    return self.title

```

Рис. 9.46. Добавление функции display\_author в модель данных Book



	НАЗВАНИЕ КНИГИ	ЖАНР КНИГИ	ЯЗЫК КНИГИ	АВТОРЫ
<input type="checkbox"/>	Продавец воздуха	Фантастика	Русский	Беляев
<input type="checkbox"/>	Дубровский	Проза	Русский	Пушкин
<input type="checkbox"/>	Война и мир	Проза	Русский	Толстой
<input type="checkbox"/>	Золотой теленок	Приключения	Русский	Ильф, Петров
<input type="checkbox"/>	12 стульев	Приключения	Русский	Ильф, Петров

Рис. 9.47. Развёрнутый список книг на основе модернизированной модели данных Book

**Листинг 9.20. Измененный код модуля admin.py**

```
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'genre', 'language', 'display_author')
```

Так как мы внесли изменения в модель, то нужно выполнить миграцию данных:

```
python manage.py makemigrations
python manage.py migrate
```

После этого необходимо перезапустить сервер и снова войти в административную панель. Если теперь нажать на ссылку со списком книг, то мы получим такую страницу (рис. 9.47).

Как видно из данного рисунка, после внесенных изменений в таблице показана более полная информация о книгах, а у книг «Золотой теленок» и «12 стульев» авторы отображены в виде списка.

### 9.7.3. Добавление фильтров к спискам

Если в окне администратора Django в списке или в таблице присутствует множество элементов, то имеется возможность отфильтровать эти данные по некоторым признакам. Это выполняется путем задания фильтра в атрибуте `list_filter`. Откроем файл `admin.py` и добавим атрибуты `list_filter` в классы `BookAdmin` (книги) и `BookInstanceAdmin` (экземпляры книги), как показано в листинге 9.21.

**Листинг 9.21. Измененный код модуля admin.py**

```
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'genre', 'language', 'display_author')
    list_filter = ('genre', 'author')
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    list_filter = ('book', 'status')
```

Здесь к классу, описывающему книги, к списку отображаемых полей добавлен фильтр на жанр и на авторов книг, а к классу, описывающему экземпляры книг, добавлен фильтр на книги и на жанр книг.

Если теперь нажать на ссылку для отображений книг (**Books**), то мы получим страницу со сведениями о книгах, на которой справа появятся два фильтра: на названия книг и на их авторов (рис. 9.48).

Если мы теперь в жанрах выберем раздел **Приключения**, а в авторах — **Ильф**, то в списке останутся две книги, которые соответствуют этим фильтрам (рис. 9.49).

Если в панели администратора нажать на ссылку со списком экземпляров книг (**Book instance**), то мы получим страницу, на которой справа появятся соответствующие фильтры по жанрам и по авторам книг (рис. 9.50).

Эти фильтры позволяют получать список экземпляров книг в различных разрезах.

Выберите book для изменения

Действие:  Выполнить

Выбрано 0 объектов из 5

<input type="checkbox"/>	НАЗВАНИЕ КНИГИ	ЖАНР КНИГИ	ЯЗЫК КНИГИ	АВТОРЫ
<input type="checkbox"/>	Продавец воздуха	Фантастика	Русский	Беляев
<input type="checkbox"/>	Дубровский	Проза	Русский	Пушкин
<input type="checkbox"/>	Война и мир	Проза	Русский	Толстой
<input type="checkbox"/>	Золотой теленок	Приключения	Русский	Ильф, Петров
<input type="checkbox"/>	12 стульев	Приключения	Русский	Ильф, Петров

5 books

**ФИЛЬТР**

↓ Жанр книги

- Все
- Детективы
- Поззия
- Приключения
- Проза
- Фантастика
- Научно-техническая
- Учебная
- 

**Фильтры**

↓ Автор книги

- Все
- Беляев
- Пушкин
- Толстой
- Ильф
- Петров

Рис. 9.48. Развёрнутый список книг с возможностью фильтрации по жанрам и авторам

Выберите book для изменения

Действие:  Выполнить

Выбрано 0 объектов из 2

<input type="checkbox"/>	НАЗВАНИЕ КНИГИ	ЖАНР КНИГИ	ЯЗЫК КНИГИ	АВТОРЫ
<input type="checkbox"/>	Золотой теленок	Приключения	Русский	Ильф, Петров
<input type="checkbox"/>	12 стульев	Приключения	Русский	Ильф, Петров

2 books

**ФИЛЬТР**

Сбросить все фильтры

↓ Жанр книги

- Все
- Детективы
- Поззия
- Приключения
- Проза
- Фантастика
- Научно-техническая
- Учебная

Рис. 9.49. Список книг с фильтром по жанрам Приключения и авторам Ильф

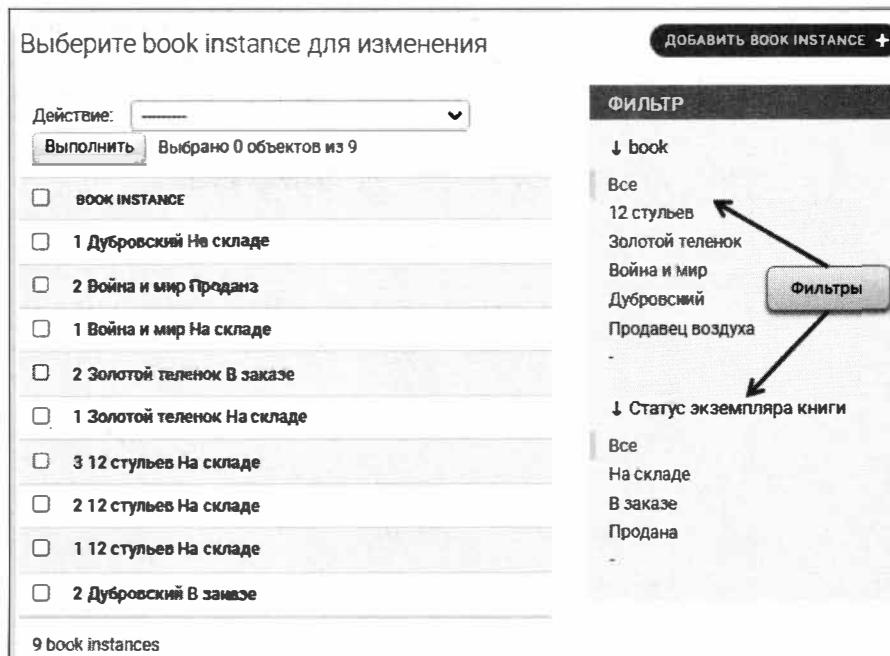


Рис. 9.50. Список экземпляров книг с фильтром по названиям книг и их статусу

#### 9.7.4. Формирование макета с подробным представлением элемента списка

По умолчанию в административной панели в списках отображаются все поля по вертикали в порядке их объявления в модели. Но можно изменить порядок их следования, указать, какие поля нужно отобразить (или исключить), отобразить поля горизонтально или вертикально и даже определить, какие виджеты редактирования использовать в формах администратора.

Модели в нашем проекте «Мир книг» достаточно просты, поэтому нет необходимости менять макет отображения полей, но мы все равно внесем некоторые изменения, чтобы показать, как это можно сделать. Обновим наш класс `AuthorAdmin` в файле `admin.py`: добавим в него атрибут `fields` и в нем укажем поля, которые должны отображаться в форме в порядке их следования. Поля в детальном отображении сведений об авторах по умолчанию будут отображаться по вертикали, но два поля (поля с датами) отобразим горизонтально, для чего дополнительно сгруппируем их в кортеже (листинг 9.22).

##### Листинг 9.22. Измененный код модуля `admin.py`

```
# Определяем класс AuthorAdmin для авторов книг
class AuthorAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name')
    fields = ['last_name', 'first_name', ('date_of_birth', 'photo')]
```

Перезагрузим наше приложение, перейдем на страницу вывода информации об авторах и получим форму в виде таблицы с горизонтальным расположением полей с усеченными сведениями об авторах: только имя и фамилия (рис. 9.51).

Выберите author для изменения		<a href="#">Добавить author +</a>
Действие:	<input type="button" value="Выполнить"/>	
Выбрано 0 объектов из 5		
<input type="checkbox"/> Имя автора	Фамилия автора	
<input type="checkbox"/> Евгений	Петров	
<input type="checkbox"/> Илья	Ильф	
<input type="checkbox"/> Лев	Толстой	
<input type="checkbox"/> Александр	Пушкин	
<input type="checkbox"/> Александр	Беляев	
5 authors		

Рис. 9.51. Усеченный список авторов книг

Теперь щелкнем мышью на фамилии одного из авторов, например **Петров**. После этого появится страница с детальной информацией об этом авторе (все поля из списка — fields). При этом поля будут расположены так, как они приведены в атрибуте fields (рис. 9.52).

Изменить author		ИСТОРИЯ
<b>Петров</b>		
Фамилия автора:	<input type="text" value="Петров"/> Введите фамилию автора	
Имя автора:	<input type="text" value="Евгений"/> Введите имя автора	
Дата рождения:	<input type="text" value="30.10.1902"/>   Сегодня	Фото автора:
	Введите дату рождения	На данный момент: Петров_Е.jpg Изменить: <input type="button" value="Выберите Файл"/> Файл не выбран
Внимание: Ваше локальное время отстает от времени сервера на 3 часа.		
<input type="button" value="Удалить"/> <input type="button" value="Сохранить и добавить другой объект"/> <input type="button" value="Сохранить и продолжить редактирование"/> <input type="button" value="СОХРАНИТЬ"/>		

Рис. 9.52. Детальная информация об авторе книг

Для изменения списка отображаемых полей также можно использовать атрибут exclude, который позволяет исключить часть полей модели из отображения в форме, при этом все остальные поля модели будут отображаться в форме.

## 9.7.5. Разделение страницы на секции с отображением связанной информации

В панели администрирования Django есть возможность разделять веб-страницы на разделы или секции (sections) для группировки связанный информации с помощью атрибута fieldsets.

Так, в модели с данными об экземплярах книг (BookInstance) мы имеем информацию о каждом конкретном экземпляре книги ( поля book, inv\_nom), а также о статусе экземпляра и дате, когда действие статуса заканчивается (status, due\_back). Мы можем разделить эти поля, разместив их в разные секции. Откроем файл admin.py и следующим образом изменим код класса BookInstanceAdmin (листинг 9.23).

### Листинг 9.23. Измененный код модуля admin.py

```
# Регистрируем класс BookInstanceAdmin для экземпляров книг
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    list_filter = ('book', 'status')
    fieldsets = (
        ('Экземпляр книги', {
            'fields': ('book', 'inv_nom')}),
        ('Статус и окончание его действия', {
            'fields': ('status', 'due_back')}),
    )
```

ID	Название	Статус	Локация
1	Дубровский	На складе	На складе
2	Золотой теленок	В заказе	На складе
3	12 стульев	На складе	На складе
4	Война и мир	Продана	На складе
5	Продавец воздуха	На складе	На складе

Рис. 9.53. Форма со списком экземпляров книг

Здесь страницу с отображением данных о конкретном экземпляре книги мы разбили на две секции. Каждая секция имеет свой заголовок 'Экземпляр книги' и 'Статус и окончание его действия' (если заголовок не нужен, то вместо него пишем `None`). В каждой секции сформирован кортеж из полей.

Перезапускаем сайт и переходим сначала к списку экземпляров книг (рис. 9.53).

Теперь на этой форме щелкнем мышью на каком-нибудь экземпляре книги. В результате откроется новое окно с подробными сведениями о выбранном экземпляре с разбивкой данных о нем на секции (рис. 9.54).

Рис. 9.54. Форма с данными об экземпляре книги с разбивкой на секции

Как видно из данного рисунка, в каждой из секций можно редактировать параметры экземпляра книги и его статус.

## 9.7.6. Встроенное редактирование связанных записей

Достаточно часто необходимо добавлять записи к связанной таблице одновременно с отображением и редактированием записи главной таблицы. Например, имеет смысл одновременно видеть как информацию о книге, так и сведения о конкретных экземплярах этой книги. Для этого необходимо объявить новый класс `inlines` и указать для него тип расположения: либо `TabularInline` (горизонтальное расположение), либо `StackedInline` (вертикальное расположение).

Открываем файл catalog/admin.py и в него добавляем код, формирующий новый класс BooksInstanceInline. Затем дописываем строку в класс BookAdmin, как это показано в листинге 9.24 (изменения выделены серым фоном).

#### Листинг 9.24. Измененный код модуля admin.py

```
class BooksInstanceInline(admin.TabularInline):
    model = BookInstance
# admin.site.register(Book)
# Регистрируем класс BookAdmin для книг
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'genre', 'language', 'display_author')
    list_filter = ('genre', 'author')
    inlines = [BooksInstanceInline]
```

Эти изменения в коде показаны на рис. 9.55.

Перезапускаем сайт и переходим сначала к списку книг (рис. 9.56).

```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
World_books > WebBooks > catalog > admin.py
Project World_books C:\Users\...
  -> venv library root
  -> WebBooks
    -> catalog
      -> migrations
      -> __init__.py
      -> admin.py <-- Current file
      -> apps.py
      -> models.py
      -> tests.py
      -> views.py
    -> media
    -> WebBooks
admin.py
17
18 class BooksInstanceInline(admin.TabularInline):
19     model = BookInstance
20
21     + +
22     # admin.site.register(Book)
23     # Регистрируем класс BookAdmin для книг
24     @admin.register(Book)
25     class BookAdmin(admin.ModelAdmin):
26         list_display = ('title', 'genre', 'language', 'display_author')
27         list_filter = ('genre', 'author')
28         inlines = [BooksInstanceInline]
```

Рис. 9.55. Объявление и использование нового класса BooksInstanceInline

Теперь на этой форме щелкнем мышью на одной из книг. В результате откроется новое окно с подробными сведениями о выбранной книге, а также будут показаны связанные записи со сведениями обо всех экземплярах этой книги. Поскольку форма достаточно большая, то продемонстрируем ее содержание в виде двух рисунков. На рис. 9.57 показана верхняя часть формы, где мы можем видеть и редактировать всю информацию о выбранной книге.

А на рис. 9.58 показана нижняя часть формы, где мы можем добавлять и изменять сведения об экземплярах этой книги.

Применительно к этой странице разработчики Django реализовали функциональный и достаточно удобный интерфейс для разработчиков сайтов. Здесь можно одновременно

Выберите book для изменения

Действие: Выполнить Выбрано 0  
объектов из 5

<input type="checkbox"/> Название книги	Жанр книги	язык книги	авторы
<input type="checkbox"/> Продавец воздуха	Фантастика	Русский	Беляев
<input type="checkbox"/> Дубровской	Проза	Русский	Пушкин
<input type="checkbox"/> Война и мир	Проза	Русский	Толстой
<input type="checkbox"/> Золотой талисман	Приключения	Русский	Ильф, Петров
<input type="checkbox"/> 12 стульев	Приключения	Русский	Ильф, Петров

5 books

ФИЛЬТР

↓ Жанр книги

- Все
- Детективы
- Поэзия
- Приключения
- Проза
- Фантастика
- Научно-техническая
- Учебная

↓ Автор книги

Все

Рис. 9.56. Форма со списком книг

Война и мир

Название книги: Война и мир  
Введите название книги

Жанр книги: Проза  
Выберите жанр для книги

Язык книги: Русский  
Выберите язык книги

Издательство: ЛитРес  
Выберите издательство

Год издания: 1985  
Введите год издания

Автор книги:  
 Беляев  
 Пушкин  
**Толстой**  
 Ильф  
 Петров  
 +  
 Выберите автора книги Удерживайте "Control" (или "Command" на Mac), чтобы выбрать несколько значений.

Аннотация книги:  
 "Война и мир". Л. Н. Толстого - книга на все времена. Как будто она существовала всегда, настолько знакомым кажется текст, едва мы открываем первые страницы романа, настолько памятны многие его эпизоды: охота и свадьба, первый бал Наташи Ростовой, лунная ночь в Оградном, князь Андрей в сражении при Аустерлице... Сцены "мирной", семейной жизни сменяются картиными, имеющими значение для хода всей мировой истории. Но для...  
 Введите краткое описание книги

ISBN книги: 978-5-389-062  
Должно содержать 13 символов

Цена (руб.): 2500,00  
Введите цену книги

Изображение обложки: На данный момент: Война\_и\_мир.jpg  
Изменить: [ Выберите файл ] Файл не выбран  
Введите изображение обложки

Рис. 9.57. Верхняя часть формы для ввода и редактирования информации о книге

BOOK INSTANCES

ИНВЕНТАРНЫЙ НОМЕР	СТАТУС ЭКЗЕМПЛЯРА КНИГИ	ДАТА ОКОНЧАНИЯ СТАТУСА	УДАЛИТЬ?
1 Война и мир На складе	На складе	Сегодня	
Внимание: Ваше локальное время отстает от времени сервера на 3 часа.			
2 Война и мир Продана	Продана	Сегодня	
Внимание: Ваше локальное время отстает от времени сервера на 3 часа.			
	—	Сегодня	
Внимание: Ваше локальное время отстает от времени сервера на 3 часа.			
	—	Сегодня	
Внимание: Ваше локальное время отстает от времени сервера на 3 часа.			
<a href="#">Добавить еще один Book instance</a>			
<a href="#">Удалить</a>	<a href="#">Сохранить и добавить другой объект</a>	<a href="#">Сохранить и продолжить редактирование</a>	<b>СОХРАНИТЬ</b>

Рис. 9.58. Нижняя часть формы для ввода и редактирования информации об экземплярах книг

модифицировать данные в главной таблице, в связанной таблице, а также во всех подключенных вспомогательных справочниках. При этом пользователи Django применяют уже готовые программные модули и не тратят время на разработку интерфейса. При активации мышью знаков автоматически появляются подсказки, которые были предусмотрены при формировании моделей данных (рис. 9.59).



Рис. 9.59. Фрагмент нижней части формы для ввода и редактирования информации об экземплярах книг с подсказкой

Рядом с полями справочников, к которым подключена родительская и дочерняя таблицы БД, имеются значки, при активации которых можно войти в режим редактирования или пополнения справочников (рис. 9.60).

Рядом с полями, которые в модели данных имеют тип «Дата», располагается значок, при активации которого автоматически появляется панель календаря, позволяющая выбрать нужную дату, при этом сохраняется режим ввода даты с клавиатуры (рис. 9.61).

Изменить book

**Война и мир**

Название книги:	Война и мир	<b>Редактировать справочники</b>
Введите название книги		
Жанр книги:	Проза	/ + ⚡
Выберите жанр для книги		
Язык книги:	Русский	/ + ⚡
Выберите язык книги		
Издательство:	ЛитРес	/ + ⚡
Выберите издательство		

Рис. 9.60. Значки для входа в режим модификации справочников



Рис. 9.61. Возможность выбора даты из встроенного календаря

Как следует из приведенных в этом разделе материалов, административная панель Django позволяет разработчикам еще на этапе проектирования сайтов вносить в БД тестовые данные и отображать их в формах административной панели. В приведенных примерах мы вводили не только символьную и цифровую информацию, но и изображения, однако в административной панели, как и в таблицах БД, смогли увидеть только имена файлов с изображениями. Настало время разобраться с тем, как Django обеспечивает прием, хранение и отображение различных файлов, например документов и изображений.

## 9.8. Работа с файлами и изображениями в административной панели Django

В примерах из предыдущего раздела были созданы две модели, в которых предусмотрено ввод и хранение изображений:

- Author — сведения об авторах с полем photo для хранения изображения (фотографии) автора книги;
- Book — сведения о книгах с полем photo для хранения изображения обложки книги.

Напомним, что в настройках проекта (файл `settings.py`) были заданы параметры для работы с файлами:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Здесь в первой строке задан URL-адрес, во второй строке путь к папке `media`, в которой будут храниться файлы пользователей.

В самих моделях `Author` и `Book` для хранения изображений было создано поле `photo` типа `ImageField`, а также указана папка, в которой будут сохранены загруженные изображения (`images`).

Если посмотреть на структуру базы данных, то можно увидеть, что поле типа `ImageField` на уровне БД представлено полем типа `varchar` (рис. 9.62).

Имя	Тип данных	Первичный ключ
1 id	integer	
2 first_name	varchar (100)	
3 last_name	varchar (100)	
4 date_of_birth	date	
5 about	text	
6 photo	varchar (100)	

Рис. 9.62. Поле `photo` в базе данных для хранения изображений

Поле `varchar` предназначено для хранения строковых данных переменной длины (по умолчанию задано максимально допустимое число символов 100). Фактически здесь хранятся только имена загруженных файлов. Сами же файлы с изображениями будут сохранены в папке `media/images` (рис. 9.63).

Таким образом, в базе данных были записаны только имена загруженных файлов, а сами файлы фактически помещены в папку проекта `media/images`. Теперь посмотрим, как можно реализовать показ загруженных изображений в формах административной панели.

В административной панели по умолчанию отображаются все поля модели данных. Но при желании можно не только изменить порядок отображения существующих полей, но и добавить новые поля, которых не было в модели, и поясняющие надписи к этим полям. Воспользуемся такой возможностью для того, чтобы в формах административной панели показать загруженные изображения. Новое поле создается в модуле `catalog/admin.py` простым добавлением нового метода к классу с описанием модели.

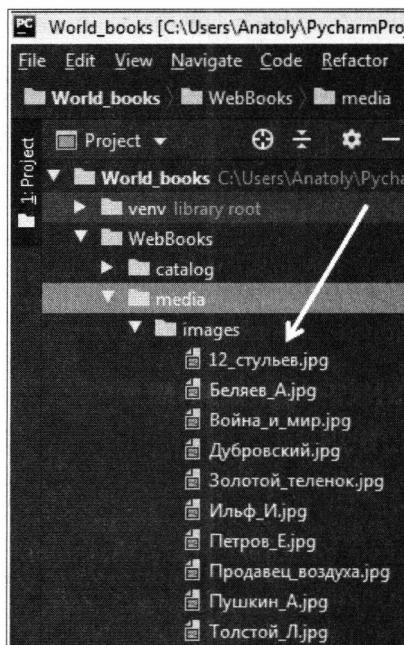


Рис. 9.63. Папка media/images с сохраненными изображениями

Открываем модуль catalog/admin.py и вносим изменения в класс AuthorAdmin (листиング 9.25, изменения выделены серым фоном).

#### Листинг 9.25. Измененный код модуля admin.py

```
# from django.utils.safestring import mark_safe
from django.utils.html import format_html
# Определяем класс AuthorAdmin для авторов книг
class AuthorAdmin(admin.ModelAdmin):
    list_display = ('first_name', 'last_name', 'photo', 'show_photo')
    fields = ['last_name', 'first_name', ('date_of_birth', 'photo')]
    readonly_fields = ["show_photo"]
    def show_photo(self, obj):
        return format_html(
            f'')
        # можно и с использованием функции mark_safe
        # return mark_safe(
        #     f'')
    show_photo.short_description = 'Фото'
```

В первой строке выполнен импорт функции Django `format_html`. Она нужна для того, чтобы отформатировать и показать изображение на HTML-странице. Затем в списке отображаемых полей `list_display` добавлено новое поле `'show_photo'`. Так как этого поля нет в модели, то мы вносим его в список полей с признаком «только для чтения» с использованием инструкции:

```
readonly_fields = ["show_photo"]
```

После этого создаем функцию (метод) с именем `show_photo()`. Этот метод в параметр `obj` принимает зарегистрированную модель с авторами книг, а затем через функцию `format_html()` возвращает ссылку на изображение `{obj.photo.url}`, а также с помощью стилей задает максимальную высоту изображения в 100 px. В последней строке создана поясняющая надпись к этому полю — «Фото».

Еще один вариант вывода изображений в административной панели предоставляет функция `mark_safe`. В листинге 9.25 в закомментированных строках показан этот вариант:

```
# from django.utils.safestring import mark_safe
# можно и с использованием функции mark_safe
#     return mark_safe(
#         f'')

```

Затем вносим аналогичные изменения в класс `BookAdmin` (листинг 9.26, изменения выделены серым фоном).

#### Листинг 9.26. Измененный код модуля `admin.py`

```
# Регистрируем класс BookAdmin для книг
@admin.register(Book)
class BookAdmin(admin.ModelAdmin):
    list_display = ('title', 'genre', 'language', 'display_author',
                    'show_photo')
    list_filter = ('genre', 'author')
    inlines = [BooksInstanceInline]
    readonly_fields = ["show_photo"]
    def show_photo(self, obj):
        return format_html(
            f'')
    show_photo.short_description = 'Обложка'
```

Здесь в списке отображаемых полей `list_display` добавлено новое поле `'show_photo'`. Затем, как и в листинге 9.25, добавлен метод для его отображения.

Осталось внести изменения в модуль `WebBooks/WebBooks/urls.py` (листинг 9.27, изменения выделены серым фоном).

#### Листинг 9.27. Измененный код модуля `urls.py`

```
from django.contrib import admin
from django.urls import path
from catalog import views
# добавлено для показа картинок
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('', views.index, name='index'),
    path('admin/', admin.site.urls),
]
```

```
if settings.DEBUG:
    if settings.DEBUG:
        urlpatterns += static(settings.MEDIA_URL,
                             document_root=settings.MEDIA_ROOT)
```

Здесь выполнен импорт модуля `settings` и переменных, определяющих пути к медиафайлам, и эти пути добавлены в диспетчер URL-адресов. Здесь используется инструкция `if settings.DEBUG`, т. е. подключение этих адресов будет происходить только тогда, когда выполняется разработка и отладка программного кода. Это означает, что указанные URL-адреса будут действовать только на этапе создания сайта на компьютере разработчика. При развертывании сайта на серверах в Интернете эти параметры нужно будет изменить.

Проверим, как это работает. Запускаем наш сервер, переходим в административную панель и нажимаем на ссылку **Authors**. После этого будет загружена форма административной панели со списком авторов (рис. 9.64).

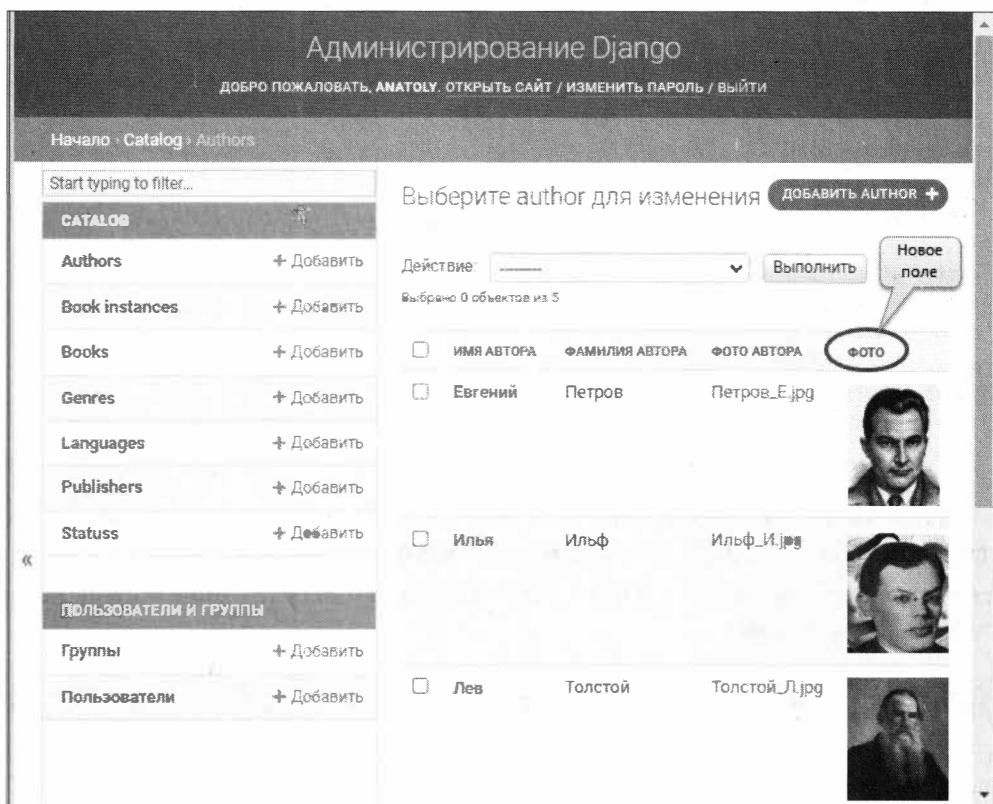


Рис. 9.64. Новое поле в административной панели для показа загруженных фото авторов книг

Как видно из данного рисунка, в списке отображаемых полей появилось новое поле **Фото**, в котором показаны загруженные изображения авторов книг. Обратите внимание, что поле **Фото автора**, в котором показано имя загруженного файла, тоже представляет собой ссылку. Если щелкнуть мышью по данной ссылке, то будет показана отдельная страница с загруженным изображением (рис. 9.65).

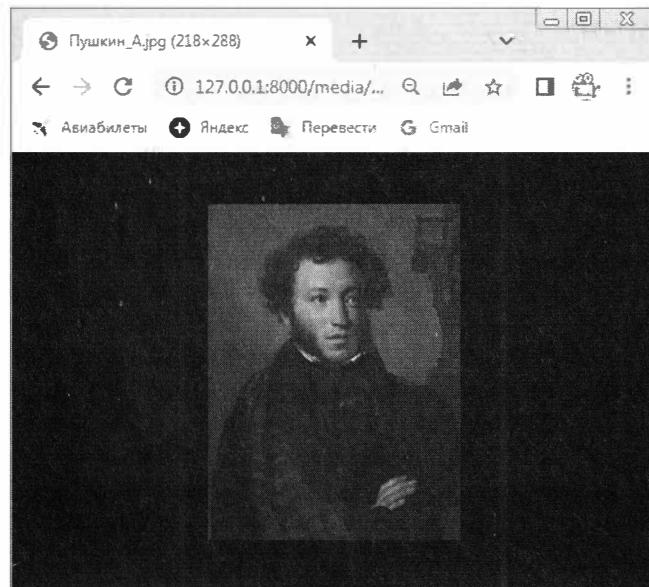


Рис. 9.65. Показ загруженных изображений из административной панели

Если теперь в административной панели нажать на ссылку **Books**, то будет загружена форма административной панели со списком книг (рис. 9.66).

Как видно из данного рисунка, в списке отображаемых полей появилось новое поле **Обложка**, в котором показаны загруженные изображения обложек книг.

A screenshot of the Django Admin interface for the 'Books' model. The top navigation bar includes 'Добавить book +'. The main area shows a table with five book entries. The first entry, 'Продавец воздуха', has its 'Обложка' field highlighted with a red oval. A callout bubble labeled 'Новое поле' points to this field. To the right of the table is a sidebar with two sections: 'ФИЛЬТР' and lists for 'Жанр книги' and 'Автор книги', each with a dropdown menu showing genre and author names respectively.

Рис. 9.66. Новое поле в административной панели для показа загруженных фото обложек книг

## 9.9. Краткие итоги

В этой главе мы разработали проект сайта и структуру таблиц для хранения данных. Затем создали модели данных и на основе этих моделей в автоматическом режиме сформировали набор соответствующих таблиц в самой БД. Мы также познакомились с возможностями административной панели Django. Узнали о настройках модуля администрирования сайта, как в самом простом виде, так и в улучшенной форме. Мы создали суперпользователя и узнали о том, как перемещаться по сайту администратора, просматривать, удалять и обновлять записи. Мы проверили работу всех моделей данных, для чего ввели сведения о нескольких книгах и их экземплярах, жанрах, об авторах, издательствах и языках.

Административная панель — это уникальный и достаточно удобный инструмент Django. Еще не было создано ни одной HTML-страницы, однако мы уже смогли заполнить БД реальными данными и протестировать ее работу. В административной панели Django есть еще опция управления правами пользователей сайта, но с этими возможностями мы познакомимся немного позже. А пока перейдем к следующей главе, которая касается создания интерфейса пользователя.



## ГЛАВА 10

# Пример создания веб-интерфейса для пользователей сайта «Мир книг»

В предыдущей главе мы рассмотрели вопросы создания структуры и моделей данных сайта «Мир книг» и использования административной панели Django для работы с информацией, которая будет храниться на сайте. Теоретически административная панель Django — это по своей сути почти готовый сайт, с которым могут работать удаленные пользователи. Однако для пользователей лучше разработать иной интерфейс. Во-первых, он должен быть более красочным и привлекательным, а во-вторых, нельзя допустить бесконтрольную возможность манипулирования информацией со стороны удаленных пользователей. Неумелыми действиями можно так испортить данные, что сайт окажется неработоспособным. Эта глава посвящена как раз теме, связанной с разработкой HTML-страниц сайта для внешних пользователей. При этом больше внимания будет уделено технологическим вопросам и меньше — дизайну разрабатываемых страниц. Тем не менее будет показано, как можно создавать привлекательный вид страниц сайта с помощью фреймворка Bootstrap.

Мы рассмотрим здесь следующие вопросы:

- формирование перечня и последовательность создания пользовательских страниц сайта;
- создание URL-преобразований (маршрутов), чтобы URL-адреса ассоциировались сервером с определенными представлениями (*views*);
- разработка представлений (*views*), чтобы на основе запросов пользователей выбирать из БД информацию, на ее основе формировать ответы и превращать их в HTML-страницы, которые будут отображаться в браузере пользователей;
- создание шаблонов страниц сайта «Мир книг»;
- возможности отображения на HTML-страницах списков и детальной информации об элементах списков.

### 10.1. Последовательность создания пользовательских страниц сайта «Мир книг»

Итак, в предыдущих главах мы создали модели данных и на их основе сформировали в БД соответствующие таблицы, которые заполнили тестовыми данными. Теперь мы готовы создать код нашей первой страницы — домашней страницы сайта «Мир книг».

Эта страница будет достаточно простой, на ней мы покажем количество записей в каждой модели и боковую навигационную панель со ссылками на другие страницы сайта (главное меню). В результате мы приобретем практический навык написания простых URL-преобразований, получения записей из базы данных, применения шаблонов для вывода данных из БД.

Вернемся к структуре приложений, написанных на Django (рис. 10.1).

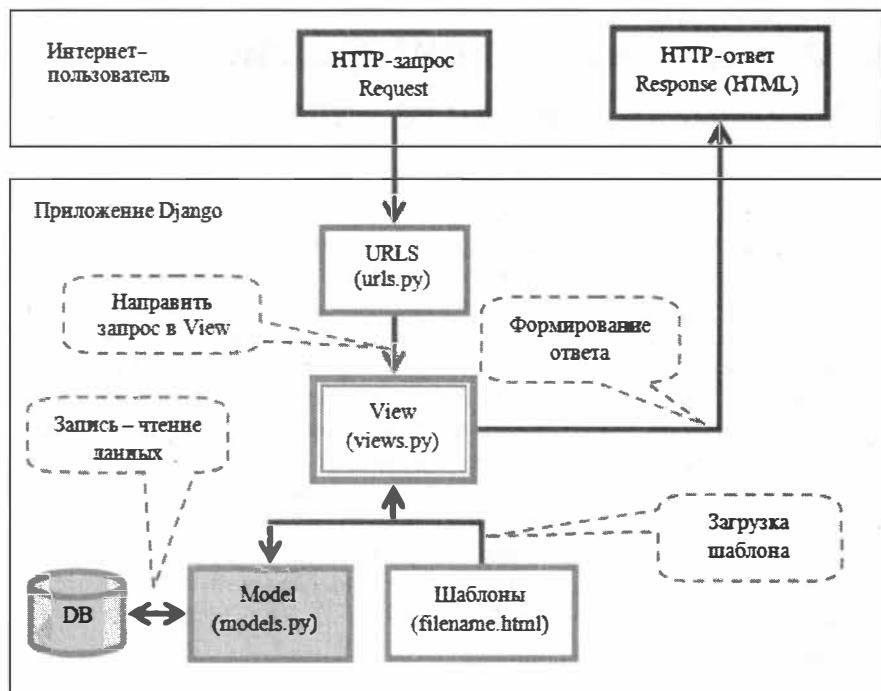


Рис. 10.1. Структура приложений, написанных на Django

Здесь в затененных квадратах показаны элементы, которые мы уже реализовали, — это модели данных и сами данные на уровне таблиц БД. Теперь пришло время написать код, который будет показывать пользователям информацию, содержащуюся в БД. Первое, что нам необходимо сделать, — это определить, какую информацию мы бы хотели показывать на наших страницах. Затем установить URL-адреса для получения соответствующих ресурсов. После чего создать URL-преобразования, представления (функции или классы) и шаблоны соответствующих страниц.

## 10.2. Определение перечня и URL-адресов страниц сайта «Мир книг»

Для конечных пользователей первая версия сайта «Мир книг» будет работать в режиме *read-only* (только для чтения). В этом случае нам нужно создать страницы, которые будут показывать списки авторов и книг, а также детальную информацию о них. Сформируем URL-адреса, которые понадобятся для наших страниц:

- index/** — домашняя (индексная) страница;
- index/books/** — список всех книг;
- index/authors/** — список всех авторов;
- index/book/<id>** — детальная информация для определенной книги со значением первичного ключа, равным **<id>**. Например: **/index/book/3** для id = 3;
- index/author/<id>** — детальная информация для определенного автора со значением первичного ключа, равным **<id>**. Например: **/index/author/11** для автора с id = 11.

Первые три URL-адреса используются для показа домашней страницы, списков книг и списка авторов книг. Содержание этих страниц будет полностью зависеть от того, что находится в базе данных.

Последние два URL-адреса служат для показа детальной информации об определенной книге или авторе. В адресе содержится соответствующее значение идентификатора (он показан как **<id>**). URL-преобразование получает эту информацию и передает ее в представление (**view**), которое применяет ее для запроса к базе данных. Для кодирования и применения этой информации в URL-адресе нам понадобится только одно URL-преобразование, соответствующее представление и шаблон страницы для показа любой книги или автора.

В Django можно конструировать URL-адреса любым удобным способом. Можно закодировать информацию в теле URL-адреса, как показано ранее, или создать URL-адрес типа GET (например: **/book/?id=6**). Независимо от способа формирования URL-адресов, они должны быть понятными, логичными и читабельными. Документация Django рекомендует кодировать информацию в теле URL-адреса — на практике это приводит к улучшению структуры сайта.

## 10.3. Создание главной страницы сайта «Мир книг»

Первой страницей, которую мы создадим, будет главная страница сайта (**index/**). Это небольшая HTML-страница, показывающая сведения о книгах, полученные из базы данных. Для того чтобы проделать эту работу, мы вначале создадим URL-преобразование, затем представление и шаблон. Рекомендуется уделить большее внимание этому разделу, поскольку описанная здесь последовательность действий пригодится при создании остальных страниц сайта. На первом шаге будут созданы представление **view** и шаблоны в минимальной конфигурации, затем будут постепенно наращиваться функциональные возможности сайта.

### 10.3.1. Создание URL-преобразования

Итак, приступим к созданию URL-преобразования. Для начала изменим базовый модуль URL-преобразований, который находится в файле **WebBooks/WebBooks/urls.py**. Напишем в нем код, приведенный в листинге 10.1.

#### Листинг 10.1. Код модуля **WebBooks/WebBooks/urls.py**

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

```
# добавлено для работы с медиафайлами
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('', include('catalog.urls')),
    path('admin/', admin.site.urls),
]
# добавлено для работы с медиафайлами локально
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                          document_root=settings.MEDIA_ROOT)
```

В этом модуле указано всего два URL-адреса:

- `path('', include('catalog.urls'))` — если в запросе пользователя указана пустая строка, то произойдет переадресация к аналогичному файлу, расположенному в приложении catalog;
- `path('admin/', admin.site.urls)` — если в запросе присутствует адрес 'admin/', то будет вызвана административная панель сайта.

Здесь также добавлены пути к медиафайлам, которые будут использоваться на этапе разработки и отладки программного обеспечения сайта.

Теперь в папке с нашим приложением catalog создадим новый файл catalog/urls.py, в котором будут указаны URL-адреса всех страниц нашего сайта (приложения catalog), и напишем в нем код листинга 10.2, как показано на рис. 10.2.

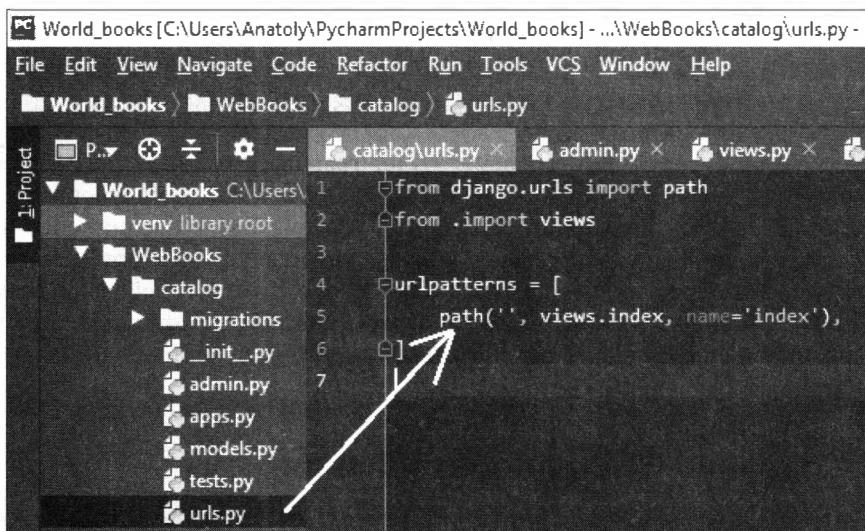


Рис. 10.2. Создание URL-преобразования для перехода на главную страницу сайта «Мир книг»

#### Листинг 10.2. Код модуля catalog/urls.py

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

```
urlpatterns = [
    path('', views.index, name='index'),
]
```

Здесь функция `path()` определяет URL-шаблон (в нашем случае это пустая строка в апострофах), функцию отображения главной страницы сайта (`views.index`) и уникальное имя для функции отображения (`name='index'`). Теперь если в запросе, поступившем от внешнего пользователя, присутствует пустая строка, то этот запрос будет адресован к функции `index()`. В свою очередь, в этой функции будет прописано обращение к БД для загрузки данных и вызов страницы, которая эти данные вернет пользователю.

В функции `path()` определен параметр `name`, который уникально объявляет, что мы имеем дело с частным URL-преобразованием. После такого объявления можно использовать это имя для «обратного» (`reverse`) преобразования, т. е. для динамического создания URL-адреса, указывающего на определенный ресурс. Например, если мы задали такое символическое имя, то теперь можно ссылаться на нашу главную страницу сайта при помощи создания следующей ссылки внутри какого-либо HTML-шаблона:

```
<a href="{% url 'index' %}">Главная страница</a>
```

Применение «обратного» URL-преобразования — это достаточно гибкий и эффективный подход для создания ссылок на различные страницы сайта. В данном случае для вызова домашней страницы в шаблоне Django достаточно указать ее имя — `{% url 'index' %}`.

С адресацией главной страницы вопрос решен, теперь нужно создать соответствующее представление (`view`).

### 10.3.2. Создание упрощенного представления (`view`)

Представление (`view`) — это программный модуль, который обрабатывает HTTP-запрос, поступивший от пользователя. Представление может быть реализовано в виде функции или класса, мы реализуем его в виде функции. В упрощенном виде в этом программном модуле сформируем текстовые переменные, загрузим их в шаблон HTML-страницы, затем вернем сгенерированную страницу пользователю в виде HTTP-ответа. Откроем файл `catalog\views.py`. В предыдущих главах мы внесли в него самый простой код, говорящий о том, что мы якобы попали на главную страницу сайта:

```
from django.http import HttpResponse

def index(request):
    return HttpResponse("Главная страница сайта Мир книг!")
```

Заменим ранее созданные программные строки функции `index()` на код листинга 10.3.

#### Листинг 10.3. Код модуля `views.py`

```
from django.shortcuts import render
def index(request):
    # Словарь для передачи данных в шаблон
    text_head = 'Это заголовок главной страницы сайта'
```

```

text_body = 'Это содержимое главной страницы сайта'
context = {'text_head': text_head, 'text_body': text_body}
# передача словаря context с данными в шаблон
return render(request, 'catalog/index.html', context)

```

В этом модуле сначала выполнен импорт функции `render` (выводить, отображать). Затем созданы две текстовые переменные `text_head` и `text_body`, их значения загружены в объект `context`, который представляет собой словарь Python. Через функцию `render` этот словарь передан в шаблон HTML-страницы `catalog/index.html`. Это пока самый простой, можно сказать, отладочный код. Нам нужно убедиться, что все URL-адреса действительны, настройки в проекте выполнены правильно и корректно подключились файлы Bootstrap.

Так как здесь выполняется обращение к шаблону с именем `index.html`, то его нужно создать. Для этого в директории `templates/catalog/` создадим файл с именем `index.html` и напишем в нем код листинга 10.4.

#### Листинг 10.4. Код шаблона templates/catalog/index.html

```

{% extends "catalog/base.html" %}
{% block title %}Мир книг{% endblock title %}
{% block header %}
    <h1>{{ text_head }}</h1>
{% endblock header %}
{% block content%}
    <div class="container-fluid my-2">
        <div class="row">
            <div class="col text-center">
                <div>{{ text_body }}</div>
            </div>
        </div>
    </div>
{% endblock content %}

```

Это достаточно упрощенный код шаблона главной страницы. В самой первой строке подключен базовый шаблон (`catalog/base.html`). Это основной шаблон, который отвечает за внешний вид всего сайта (он будет создан на следующем шаге). Далее идут блоки, заключенные в теги Django. В блоке `title` выводится текст, который будет виден в окне браузера в заголовке HTML-страницы. Далее в блоке `header` в теге `{{ text_head }}` будет выводиться текст, переданный из представления `view`. Этот текст будет находиться в самой верхней части главной страницы, это, по сути, ее заголовок. Далее следует блок `content` — основное содержание страницы. Здесь могут быть таблицы, изображения, формы и т. п. Пока мы максимально упростили код этого блока, в котором присутствует адаптивный контейнер Bootstrap (`class="container-fluid my-2"`) с одной строкой (`class="row"`) и встроенной в нее колонкой (`class="col text-center"`). В этой колонке в теге `{{ text_body }}` будет выведен текст, который передан из представления `view`.

Поскольку в первой строке подключен базовый шаблон, то его тоже необходимо создать. В папке с шаблонами создаем файл `templates/catalog/base.html`. В этом шаблоне напишем код листинга 10.5.

**Листинг 10.5. Код шаблона templates/catalog/base.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
    {%- load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <div class="container-fluid p-1 bg-primary text-white text-center">
        <div class="row">
            <div class="col-2 text-start">
                
            </div>
            <div class="col-10 ">
                <h1>Это шапка сайта с логотипом</h1>
            </div>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row bg-warning text-center">
            <h6>
                <a href="{% url 'index' %}">Главная страница</a>
                <a href="{% url 'index' %}">О компании</a>
                <a href="{% url 'index' %}">Контакты</a>
            </h6>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row text-center text-primary fs-1 fw-bold">
            <div>{%- block header %}<br>
                {% endblock header %}<br>
            </div>
        </div>
        <div class="row text-center text-body">
            <div class="col-2 ">
                {% block sidebar %}<br>
                <nav class="nav flex-column">
                    <a class="nav-link" href="{% url 'index' %}">Ссылка 1</a>
                    <a class="nav-link" href="{% url 'index' %}">Ссылка 2</a>
                    <a class="nav-link" href="{% url 'index' %}">Ссылка 3</a>
                </nav>
                {% endblock sidebar %}<br>
            </div>
            <div class="col-10 " >
                <div class="row text-center">
                    {% block content %}<br>
                    {% endblock content %}<br>
                </div>
            </div>
        </div>
    </div>
```

```

        </div>
    </div>
</div>
<div class="container-fluid">
    <div class="row bg-primary text-center text-white lh-1">
        {% block footer %}
            <p>Это подвал (footer) всех страниц сайта</p>
        {% endblock footer%}
    </div>
</div>
</body>
</html>

```

Поскольку это достаточно важный элемент нашего сайта, подробно проанализируем данный код.

В самом начале находится блок `head` — заголовок страницы:

```

<head>
    {% load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
</head>

```

В этом блоке через тег Django `{% load static %}` сделана ссылка на папку `static`, в которой содержатся статические файлы. В эту папку мы поместили изображения и файлы фреймворка Bootstrap. В этом же блоке файлы Bootstrap подключены к базовому шаблону нашего сайта.

Далее следует тег `body` — тело HTML-страницы. В этом блоке создано четыре адаптивных контейнера Bootstrap `class="container-fluid"` для размещения:

- шапки сайта;
- навигационной панели с главным (верхним) меню сайта;
- основного контента;
- «подвала» страниц сайта.

В первом контейнере находится, так называемая шапка сайта. Этот контейнер имеет одну строку и две колонки. В первой колонке выведен логотип сайта (это изображение из папки `static`) и строка с названием сайта.

В втором контейнере создана одна строка, в которой находится навигационная панель сайта. В этой строке пока мы поместили ссылки на три страницы сайта: «Главная страница», «О компании», «Контакты». Так как страницы «О компании», «Контакты» еще не созданы, то для них временно указаны адреса главной страницы — `{% url 'index' %}`.

В третьем контейнере созданы две строки, при этом вторая строка разбита на две колонки. В первой строке находится блок для размещения заголовка страницы — `{% block header%}`. В первой колонке второй строки находится блок «боковая панель» — `{% block sidebar %}`, в которой будет выводиться дополнительное (боковое) меню сайта. Во вто-

рой колонке второй строки находится блок с основным контентом страницы — {`% block content%`}, в котором будет выводиться основное содержание страницы сайта.

В четвертом контейнере создана одна строка, которая представляет собой «подвал» (`footer`) страницы — {`% block footer %`}.

Схематично структура базового шаблона показана на рис. 10.3.

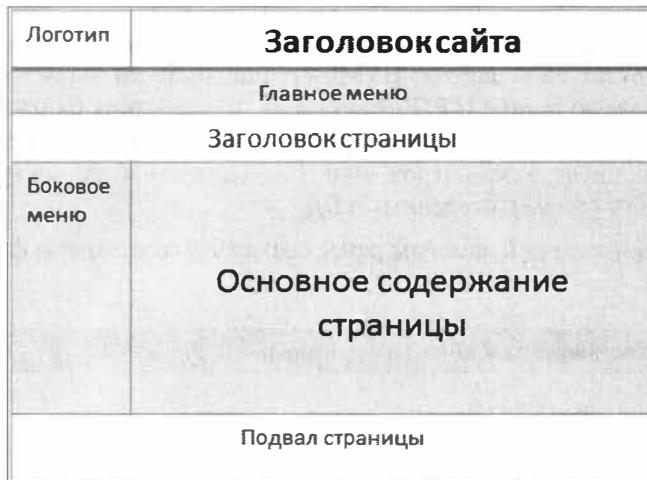


Рис. 10.3. Структура базового шаблона сайта

Итак, мы создали все базовые элементы для того, чтобы проверить в работе макет созданного сайта. Разместим в папке `static/images/` файл с изображением логотипа сайта и запустим наш веб-сервер, выполнив команду

```
python manage.py runserver
```

Главная страница сайта будет выглядеть так, как показано на рис. 10.4.

Как видно из данного рисунка, в паре шаблонов `base.html` и `index.html` появилось содержимое двух текстовых переменных, переданных из представления `views.index`: «Это заголовок главной страницы сайта», «Это содержимое главной страницы сайта».

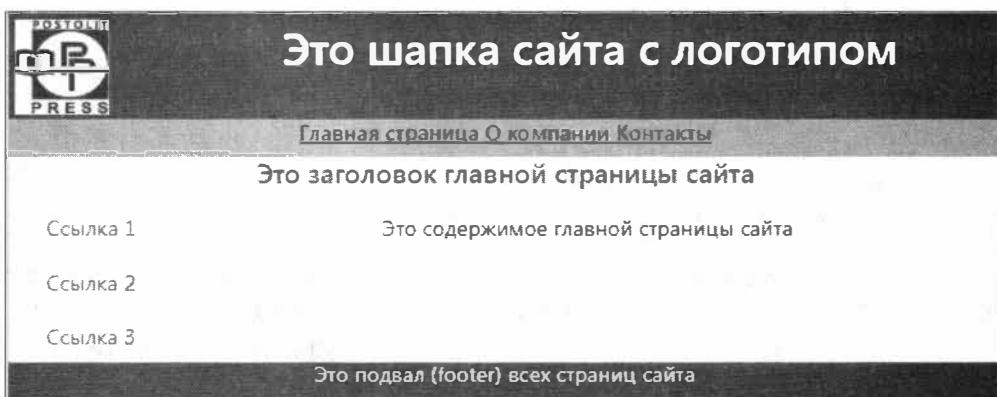


Рис. 10.4. Внешний вид главной страницы сайта `index.html` на основе базового шаблона `base.html`

Итак, базовые элементы сайта созданы, теперь можно заниматься модификацией программного кода, созданием новых страниц, улучшая как функциональные возможности сайта, так и его интерфейс.

### 10.3.3. Изменение представления (*view*) главной страницы сайта

В нашем проекте программный код главной страницы сайта получит информацию из базы данных, загрузит ее в шаблон HTML-страницы, затем вернет сгенерированную страницу пользователю в виде HTTP-ответа. Все эти действия будет выполнять функция с именем `index()`. На этой странице реализуем функционал чтения из БД данных о книгах (`Book`), о числе экземпляров книг (`BookInstance`) и их доступности, а также о числе авторов книг (`Author`), введенных в БД.

Откроем файл `catalog/views.py` и заменим ранее созданное содержимое функции `index()` на код листинга 10.6.

#### Листинг 10.6. Измененный код файла `catalog/views.py`

```
from django.shortcuts import render
from .models import Book, Author, BookInstance

def index(request):
    text_head = 'На нашем сайте вы можете получить книги в электронном виде!'
    # Данные о книгах и их количестве
    books = Book.objects.all()
    num_books = Book.objects.all().count()
    # Данные об экземплярах книг в БД
    num_instances = BookInstance.objects.all().count()
    # Доступные книги (статус = 'На складе')
    num_instances_available = BookInstance.objects.filter(
        status__exact=2).count()
    # Данные об авторах книг
    authors = Author.objects
    num_authors = Author.objects.count()
    # Словарь для передачи данных в шаблон index.html
    context = {'text_head': text_head,
               'books': books, 'num_books': num_books,
               'num_instances': num_instances,
               'num_instances_available': num_instances_available,
               'authors': authors, 'num_authors': num_authors}
    # передача словаря context с данными в шаблон
    return render(request, 'catalog/index.html', context)
```

Во второй строке этого программного модуля был выполнен импорт следующих моделей: `Book` (книги), `Author` (авторы), `BookInstance` (экземпляры книг).

В первой части функции `index()` выполняются запросы к БД и выборка данных:

- о книгах — `books`;
- о количестве книг — `num_books`;

- о числе экземпляров книг — num\_instances;
- о числе экземпляров книг, доступных к продаже (находятся на складе) — num\_instances\_available;
- об авторах книг — authors;
- о числе авторов книг — num\_authors.

Во второй части функции index() создается словарь context, в который загружаются все данные, полученные из БД. Затем вызывается функция render(), которая в качестве ответа пользователю создает и возвращает HTML-страницу index.html. Функция render имеет следующие аргументы: объект request (запрос типа HttpRequest), шаблон HTML-страницы (catalog/index.html), а также объект context, представляющий собой словарь Python с полученными из БД данными.

Теперь нужно изменить шаблон HTML-страницы, в которой пользователь получит ответ на свой запрос.

### 10.3.4. Модификация шаблона главной страницы сайта «Мир книг»

Теперь мы можем перейти к внесению изменений в главную страницу нашего сайта (ее еще называют *домашней*). Мы уже создали заготовку для этой страницы в предыдущих разделах — файл \WebBooks\templates\catalog\index.html. Внесем в него код листинга 10.7.

#### Листинг 10.7. Измененный код файла \WebBooks\templates\catalog\index.html

```
% extends "catalog/base.html" %}
% block title %}Мир книг% endblock title %}
% block header %
<h5>{{ text_head }}</h5>
% endblock header %
% block content%
<div class="container-fluid my-2">
<div class="row bg-success text-white mx-2">
<div class="col text-center">
<p>Добро пожаловать на сайт <em>Мир книг</em>. Это очень
простой веб-сайт, написанный на Django и Bootstrap.
Разработан на Python в качестве учебного примера
с использованием PyCharm.</p>
</div>
</div>
<div class="row text-center text-dark lh-2">
<h4>Сведения из базы данных</h4>
</div>
<div class="row my-2 text-white">
<div class="col mx-2 bg-primary">Количество книг -
{{ num_books }}</div>
<div class="col mx-2 bg-primary">На складе в наличии -
{{ num_instances }}</div>
```

```

<div class="col mx-2 bg-primary">Количество авторов -
    {{num_authors}}</div>
</div>
<div class="row text-center text-dark lh-2">
    <h3>Книги</h3>
</div>
{% if num_books > 0 %}
    <div class="row my-2">
        {% for obj in books %}
            <div class="card" style="width: 9rem;">
                
                <div class="card-body">
                    <p class="card-text small">
                        {{obj.title}}, Цена:{{obj.price}} руб.
                    </p>
                </div>
            </div>
        {% endfor %}
    {% endif %}
</div>
{% endblock content %}

```

Как можно видеть из первой строки данного листинга, эта страница является расширением базового шаблона `base.html`.

После этого следует блок `header` — заголовок данной страницы `{% block header %}`. В этом заголовке выводится текст, который передан в данный шаблон в переменной `text_head` из представления `views.index`.

Затем следует блок `content`. Его содержимое будет полностью перенесено в соответствующий блок `content`, который находится в базовом шаблоне. Макет этого блока сформирован на основе адаптивных классов и тегов фреймворка Bootstrap и состоит из следующих разделов:

- вывод текстовой информации;
- вывод сведений о числе элементов в базе данных (число книг, число экземпляров книг, число авторов);
- вывод карточек с изображением книг.
- Для вывода сведений о числе элементов в базе данных сформированы три адаптивных блока, которые представляют три колонки одной строки. Эти блоки постоянно присутствуют на странице, в них меняются только числовые данные, полученные из БД.
- Для вывода карточек с изображением книг используются специальные теги Django:
  - `{% if num_books > 0 %}` — проверяет наличие книг в БД;
  - `{% for obj in books %}` — организует цикл перебора найденных книг.
- В блоке `if` проверяется условие — есть ли в БД записи о книгах. Если число найденных в БД книг больше нуля, то в цикле выполняется выбор всех найденных книг и формирование карточки по каждой книге. Карточки создаются на основе класса

class="card" фреймворка Bootstrap. В карточку заносится: изображение обложки книги — {{obj.photo.url}}, название книги — {{obj.title}}, цена книги — {{obj.price}}.

Важно отметить, что имена переменных в шаблоне HTML-страницы должны совпадать с именами передаваемых ключей из словаря context, который передается из представления (view) через функцию render().

Теперь немного изменим код базового шаблона base.html (листинг 10.8, изменения выделены серым фоном).

#### Листинг 10.8. Измененный код файла templates\catalog\base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <div class="container-fluid p-1 bg-primary text-white text-center">
        <div class="row">
            <div class="col-2 text-start">
                
            </div>
            <div class="col-10 ">
                <h3>Мир книг – печатные и электронные интерактивные книги</h3>
            </div>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row bg-warning text-center">
            <h6>
                <a href="{% url 'index' %}">Главная страница</a>
                <a href="{% url 'index' %}">О компании</a>
                <a href="{% url 'index' %}">Контакты</a>
            </h6>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row text-center text-primary fs-1 fw-bold">
            <div>{% block header%}
                {% endblock header %}
            </div>
        </div>
        <div class="row text-start text-body">
            <div class="col-2 ">
                {% block sidebar %}

```

```
<nav class="nav flex-column">
    <a class="nav-link" href="{% url 'index' %}">Все книги</a>
    <a class="nav-link" href="{% url 'index' %}">Все авторы</a>
</nav>
{%
    endblock sidebar
}
</div>
<div class="col-10" >
    <div class="row text-center">
        {%
            block content%
        }
        {%
            endblock content %
        }
    </div>
</div>
</div>
<div class="container-fluid">
    <div class="row bg-info text-center text-dark small lh-sm pt-2">
        {%
            block footer %
        }
        <p>Copyright ООО "Интеллектуальные информационные системы", 2023.
            Все права защищены</p>
        {%
            endblock footer%
        }
    </div>
</div>
</body>
</html>
```

В этом файле мы изменили немногое:

- надпись в шапке сайта (в строке с логотипом);
- опции меню в боковой панели;
- надпись в строке с «подвалом» страницы (footer);
- цвета в некоторых блоках.

К настоящему моменту сделано все необходимое, чтобы показать главную страницу нашего сайта. Запускаем веб-сервер командой

```
python manage.py runserver
```

Вводим в адресной строке браузера адрес <http://127.0.0.1:8000/>. Если все настроено правильно, наш сайт должен выглядеть так, как показано на рис. 10.5.

Как видно из данного рисунка, на главной странице сайта выведена та информация о книгах, которая была введена в БД через панель администратора.

Сейчас у нас нет возможности воспользоваться ссылками на страницы **Все книги** и **Все авторы**, потому что URL-адреса, представления и шаблоны для этих страниц еще не созданы. Мы просто объявили метки для соответствующих ссылок в базовом шаблоне `base.html`, но сами шаблоны страниц и соответствующие представления для них пока не создали. Тем не менее мы сформировали домашнюю страницу для нашего сайта, которая показывает число некоторых записей в базе данных, сведения о книгах и содержит ссылки на другие, еще не созданные страницы.

Следует отметить, что главная страница сайта — это динамическая страница, она станет автоматически обновляться по мере того, как будет меняться содержимое БД.

**Мир книг - печатные и электронные интерактивные книги**

Главная страница О компании Контакты

На нашем сайте вы можете получить книги в электронном виде

Все книги      Добро пожаловать на сайт *Mir книг*. Это очень простой веб-сайт, написанный на Django и Bootstrap. Разработан на Python в качестве учебного примера с использованием PyCharm.

Все авторы

**Сведения из базы данных**

Количество книг - 5      На складе в наличии - 9      Количество авторов - 5

### Книги

12 стульев, Цена:220,00 руб.	Золотой теленок, Цена:210,00 руб.	Война и мир, Цена:2500,00 руб.	Дубровский, Цена:350,00 руб.	Продавец воздуха, Цена:250,00 руб.

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены

Рис. 10.5. Главная страница сайта «Мир книг»

**Мир книг - печатные и электронные  
интерактивные книги**

Главная страница О компании Контакты

На нашем сайте вы можете получить книги в  
электронном виде

Все книги      Добро пожаловать на сайт *Mir книг*. Это очень  
простой веб-сайт, написанный на Django и  
Bootstrap. Разработан на Python в качестве  
учебного примера с использованием PyCharm.

Все авторы

**Сведения из базы данных**

Количество книг - 5      На складе в наличии - 9      Количество авторов - 5

### Книги

12 стульев, Цена:220,00 руб.	Золотой теленок, Цена:210,00 руб.	Война и мир, Цена:2500,00 руб.

Рис. 10.6. Главная страница  
сайта в формате  
мобильного телефона

Кроме того, это адаптивная страница, т. к. при ее макетировании использовались элементы фреймворка Bootstrap. Если эту страницу загрузить на мобильный телефон, то адаптивные блоки поменяют свое расположение, при этом текстовая и графическая информация будет отображаться в удобно читаемом масштабе (рис. 10.6).

В следующих разделах мы создадим недостающие страницы сайта. А пока рассмотрим возможность отображения детальной информации об элементе списка на примере списка книг, т. е. разберемся, как, имея список книг, получить подробные сведения о каждой книге.

## 10.4. Создание страницы со списком книг на основе класса *ListView*

В этом разделе мы продолжим разработку сайта «Мир книг», добавляя в него новые страницы, в частности создадим страницу, в которой будет показан список всех книг, находящихся в БД. В предыдущем разделе представление (*view*) для формирования главной страницы было реализовано на основе функции `index()`.

Здесь мы познакомимся с еще одной возможностью визуализации данных на основе классов

CBV (Class-Based Views). В Django имеется группа встроенных классов для представлений (*view*), например:

- `ListView` — просмотр списка объектов;
- `CreateView` — создание конкретного объекта;
- `FormView` — отображение формы;
- `DetailView` — просмотр определенного объекта;
- `UpdateView` — обновление конкретного объекта;
- `DeleteView` — удаление конкретного объекта и т. п.

Познакомиться с этими классами можно в оригинальной документации, а в данном разделе мы будем использовать встроенные в Django базовые классы `ListView` (для визуализации списка книг) и `DetailView` (для визуализации детальной информации о конкретной книге). Чаще всего именно эти классы применяют на практике, т. к. объектно-ориентированный подход позволяет заметно сократить и упростить программный код, который необходим для формирования HTML-страниц.

Также будет продемонстрирована возможность постраничного показа данных (*pagination*) в тех случаях, когда выводимый список достаточно большой и занимает много места на отображаемой HTML-странице.

Итак, приступим к созданию страницы с отображением списка книг, имеющихся в БД, которая будет доступна по адресу `catalog/books/`.

На этой странице в каждой записи будет показано название книги, автор, жанр и изображение обложки, при этом каждое название книги будет являться гиперссылкой, нажав на которую пользователь перейдет на страницу с подробной информации об этой книге. Наша страница будет иметь ту же структуру, что и все остальные страницы сай-

та. Для ее создания мы воспользуемся базовым шаблоном сайта (`base.html`), который был рассмотрен в предыдущем разделе.

И начнем мы с сопоставления URL-адресов. Для этого откроем файл `\WebBooks\catalog\urls.py` и добавим в него строки, выделенные в листинге 10.9 серым фоном.

#### Листинг 10.9. Измененный код файла `\WebBooks\catalog\urls.py`

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view() name='books'),
]
```

Практически так же, как и для главной страницы сайта, функция `path()` связывает маршрут (URL-адрес) страницы (`books/`) с классом в представлении `views.BookListView`. `as_view()`. Кроме того, здесь определено имя для ссылки на данную страницу сайта (`name='books'`). Обратите внимание, метод `as_view()` нужно вызвать, т. е. поставить в конце круглые скобки, а не просто передать ссылку на него. В результате выполняется вся запрограммированная работа по созданию экземпляра класса и гарантируется вызов правильных методов для входящих HTTP-запросов от пользователей.

Конечно, мы могли бы реализовать показ списка книг при помощи обычной функции, как это делалось для главной страницы сайта. Иначе говоря, выполнить запрос на получение списка всех книг из базы данных, после чего вызвать функцию `render()`, которой передать этот список, и отправить его в соответствующий шаблон страницы. Но здесь мы воспользуемся классом, который будет наследником существующего в Django класса `ListView` для отображения списка данных. В классе `ListView` уже реализован функционал для того, чтобы получить и показать список книг, а значит, мы справимся со своей задачей меньшим числом строк программного кода.

Откроем файл `catalog\views.py` и добавим в него класс `BookListView` с помощью кода листинга 10.10 (изменения выделены серым фоном).

#### Листинг 10.10. Измененный код файла `catalog\views.py`

```
from .models import Book, Author, BookInstance
from django.views.generic import ListView
class BookListView(ListView):
    model = Book
    context_object_name = 'books'
```

Вот, собственно, и все, это весь код!

Здесь мы импортировали базовый класс (`import ListView`) и на его основе создали собственный класс `BookListView(ListView)`. Можно было бы обойтись всего одной строкой кода — `model = Book` (подключением модели данных `Book`). В созданном классе по умолчанию будет сформирован ряд объектов, в частности:

- `object_list` — список с данными о книгах, полученных из БД;
- `template_name` — имя шаблона для вывода списка книг (по умолчанию `book_list.html`).

Можно воспользоваться этими объектами, а при желании можно переопределить эти имена, например:

```
context_object_name = 'books'  
template_name = "my_books.html"
```

Мы оставили имя шаблона для вывода списка книг без изменения (`book_list.html`), а списку с книгами присвоили новое имя '`books`'. В итоге созданный нами класс `BookListView` выполнит запрос к базе данных, получит все записи о книгах из модели (`Book`), а затем передаст эти данные в шаблон `book_list.html`.

В этом шаблоне теперь из списка `books` можно с использованием цикла получить данные по любой книге: `books.title` — название книги, `books.genre` — жанр, `books.year` — год издания и т. д.

Теперь нужно создать шаблон для выдачи пользователю списка книг с именем `book_list.html` и поместить его по следующему пути:

`\WebBooks\templates\catalog\book_list.html`.

В этом шаблоне напишем код, приведенный в листинге 10.11.

#### Листинг 10.11. Измененный код файла `\WebBooks\templates\catalog\book_list.html`

```
{% extends "catalog/base.html" %}  
{% block content %}  
    <div class="row text-center text-dark lh-2">  
        <h4>Список книг в БД</h4>  
    </div>  
    {% if books %}  
        <table class="table table-striped table-bordered text-start">  
            <thead>  
                <tr>  
                    <th>Название</th>  
                    <th>Автор</th>  
                    <th>Жанр</th>  
                    <th>Обложка</th>  
                    <th>Показать обложку</th>  
                </tr>  
            </thead>  
            <tbody>  
                {% for book in books %}  
                    <tr>  
                        <td><a href="{{ book.id }}">{{ book.title }} </a></td>  
                        <td>{{ book.display_author }}</td>  
                        <td>{{ book.genre }}</td>  
                        <td></td>  
                        <td><a href="{{ book.photo.url }}"  
                           class="btn btn-primary"  
                           target="_blank"> Показать </a></td>  
                    </tr>  
                {% endfor %}  
            </tbody>  
        </table>
```

```
{% else %}  
    <p>В базе данных нет книг</p>  
{% endif %}  
{% endblock %}
```

Здесь, как и в шаблоне главной страницы, в первой строке мы расширяем наш базовый шаблон — {% extends "catalog/base.html" %}, а затем определяем блок с именем content. Внутри этого блока в цикле формируется и выдается список книг.

Сначала с использованием инструкции if проверяется, есть ли в БД записи с информацией о книгах — {% if books %}. Если в БД есть хотя бы одна запись о книге, то создается таблица Bootstrap, если в БД нет записей о книгах, то пользователю будет выдано сообщение «В базе данных нет книг».

В теге Bootstrap `<table>` создано пять столбцов. Сначала в теге `<thead>` формируется «шапка» таблицы, а затем в цикле {% for book in books %} столбцы таблицы заполняются следующей информацией:

- {{ book.title }} — название книги;
- {{ book.display\_author }} — автор (авторы) книги;
- {{ book.genre }} — жанр книги;
- {{ book.photo.url }} — изображение обложки книги;
- {{ book.photo.url }} — ссылка на изображение обложки книги.

Код внутри цикла представлен следующей инструкцией вывода названия книг:

```
<a href="{{ book.id }}">{{ book.title }}</a>
```

Здесь текст с названием книги представляет собой ссылку для перехода на страницу с полной информацией об этой книге. В качестве параметра в URL-адресе передается идентификатор книги в БД (book.id).

При выводе изображения обложки книги задано ограничение изображения по высоте:

```
style="max-height:100px"
```

Таким образом, независимо от размера исходных изображений, все обложки будут выведены в едином масштабе. Для того чтобы можно было посмотреть на обложку в полном размере, в последней колонке выведена кнопка со ссылкой на файл с изображением. Нажав на нее, пользователь может в отдельной вкладке браузера увидеть обложку в полноформатном виде.

Теперь нужно открыть файл с базовым шаблоном по пути

```
\WebBooks\templates\catalog\base.html
```

и указать URL-ссылку для пункта бокового меню Все книги, как показано в листинге 10.12 (изменения выделены серым фоном).

#### Листинг 10.12. Измененный код файла \WebBooks\templates\catalog\base.html

```
{% block sidebar %}  
<nav class="nav flex-column">  
    <a class="nav-link" href="{% url 'books' %}">Все книги</a>
```

```
<a class="nav-link" href="{% url 'index' %}>Все авторы</a>
</nav>
{% endblock sidebar %}
```

В этой строке мы указали, что при выборе опции меню **Все книги** будет выполнен переход к URL-преобразованию с именем 'books', а в нем происходит обращение к классу

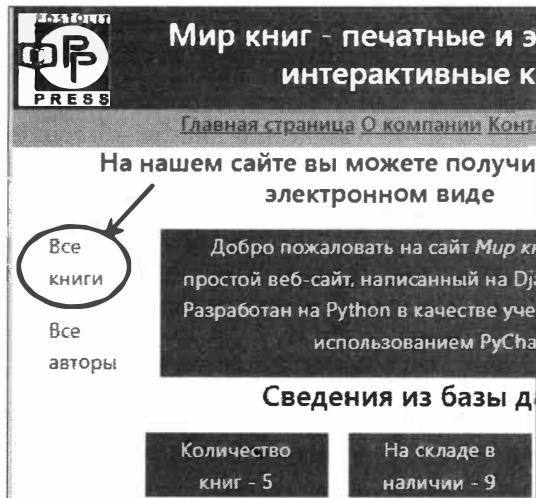


Рис. 10.7. Переход к странице Все книги на главной странице сайта «Мир книг»

Список книг в БД					
	Название	Автор	Жанр	Обложка	Показать обложку
Все книги	<a href="#">12 стульев</a>	Ильф, Петров	Приключения		<a href="#">Показать</a>
Все авторы	<a href="#">Золотой теленок</a>	Ильф, Петров	Приключения		<a href="#">Показать</a>
	<a href="#">Война и мир</a>	Толстой	Проза		<a href="#">Показать</a>
	<a href="#">Дубровский</a>	Пушкин	Проза		<a href="#">Показать</a>

Рис. 10.8. Страница Все книги сайта «Мир книг»

BookListView, который и выполнит всю работу по заполнению шаблона book\_list.html и его отправке пользователю.

Посмотрим, что у нас получилось. Запускаем наш веб-сервер, выполнив следующую команду:

```
python manage.py runserver
```

Теперь на главной странице нужно щелкнуть левой кнопкой мыши на ссылке **Все книги** в боковом меню (рис. 10.7).

Если все сделано правильно, то будет выдана страница с полным списком книг, которые занесены в БД (рис. 10.8).

На этой странице названия книг показаны в синем цвете, а это признак того, что они являются ссылками для перехода на страницу с детальной информацией о книге. Однако если щелкнуть мышью на такой ссылке (на название книги), то ничего не произойдет, поскольку мы еще не создали страницу для показа полной (детальной) информации о книге. Приступим к этому процессу.

## 10.5. Создание страницы с детальной информацией о книге на основе класса *DetailView*

Доступ к странице с детальной информацией о книге осуществляется при помощи URL-адреса catalog/book/<id> (<id> — первичный ключ в БД для этой книги). В детальную информацию о книге в дополнение к полям модели Book (автор, краткое содержание, ISBN, язык и жанр) мы также добавим информацию о доступных экземплярах книги (BookInstances), включая их статус, ожидаемой дате изменения статуса, издаельстве (publisher) и инвентарном номере (inv\_nom). Эта информация должна позволить пользователям не просто получить сведения о книге, но и убедиться, имеется ли она в наличии и является ли доступной.

Откроем файл WebBooks\catalog\urls.py и добавим строку в URL-преобразование, как показано в листинге 10.13 (изменения выделены серым фоном).

### Листинг 10.13. Измененный код файла WebBooks\catalog\urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
]
```

Практически так же, как и для страницы сайта books, функция path() связывает маршрут (URL-адрес) страницы (books/<int:pk>) с классом в представлении views.BookDetailView.as\_view(). Кроме того, здесь определено имя для ссылки на данную страницу сайта (name='book-detail').

Поскольку мы заранее не знаем, о какой книге будет запрошена информация, то в этом случае необходимо использовать функцию path с параметрами формирования URL-адреса ('books/<int:pk>/'). Здесь угловые скобки <..> необходимы для того, чтобы получить значение из URL-адреса. Преобразователь int обеспечивает гарантированное получение целочисленного параметра, а параметр pk (primary key — первичный ключ) служит для получения идентификатора книги из БД.

Здесь мы воспользуемся классом, который наследуется от существующего в Django класса DetailView (отображение элемента из списка данных). В классе DetailView уже реализован функционал для того, чтобы получить и показать информацию о конкретной книге, а значит, мы справимся со своей задачей меньшим числом строк программного кода.

Откроем файл catalog\views.py и добавим в него класс BookDetailView с помощью кода листинга 10.14 (изменения выделены серым фоном).

#### Листинг 10.14. Измененный код файла \WebBooks\catalog\views.py

```
from .models import Book, Author, BookInstance
from django.views.generic import ListView, DetailView
class BookDetailView(DetailView):
    model = Book
    context_object_name = 'book'
```

Вот, собственно, и весь код!

Здесь мы импортировали базовый класс (`import DetailView`) и на его основе создали собственный класс `BookDetailView(DetailView)`. Можно было бы обойтись всего одной строкой кода — `model = Book` (подключением модели данных `Book`). В созданном классе по умолчанию будет сформирован ряд объектов, в частности:

- `object` — данные о конкретной книге, полученные из БД;
- `template_name` — имя шаблона для вывода информации о конкретной книге (по умолчанию `book_detail.html`).

Можно воспользоваться этими объектами, а при желании можно переопределить эти имена, например:

```
context_object_name = 'book'
template_name = "my_book.html"
```

Мы оставили имя шаблона для вывода информации о книге без изменения (`book_detail.html`), а объекту с данными о конкретной книге присвоили новое имя `'book'`. В итоге созданный нами класс `DetailView` выполнит запрос к базе данных, получит данные из БД о конкретной книге, а затем передаст эти данные в шаблон `book_detail.html`. В этом шаблоне теперь из объекта `book` можно получить данные по конкретной книге: `book.title` — название книги, `book.genre` — жанр, `book.year` — год издания и т. д.

Теперь нужно создать шаблон для выдачи пользователю данных о книге с именем `book_detail.html` и поместить его по следующему пути:

`\WebBooks\templates\catalog\book_detail.html`.

В этом шаблоне напишем код, приведенный в листинге 10.15.

**Листинг 10.15. Измененный код файла \WebBooks\templates\catalog\book\_detail.html**

```
{% extends "catalog/base.html" %}  
{% block content %}  
<div class="container text-start">  
    <div class="row my-2">  
        <h1>Название книги - "{{ book.title }}"</h1>  
    </div>  
    <div class="row my-2">  
        <p><strong>Автор:</strong>  
        {% for author in book.author.all %}  
            <a href="">{{ author.first_name }} {{author.last_name}}</a>  
        {% endfor %}  
        </div>  
          
        <a href="{{ book.photo.url }}"  
               class="btn btn-primary"  
               target="_blank"> Показать</a>  
    <div class="row my-2">  
        <div class="col">  
            <p><strong>Цена:</strong> {{ book.price }} руб.</p>  
        </div>  
    </div>  
    <div class="row my-2">  
        <p><strong>Аннотация:</strong> {{ book.summary }}</p>  
    </div>  
    <div class="row my-2">  
        <div class="col mx-2 bg-primary text-white">  
            <p><strong>Жанр:</strong> {{ book.genre }}</p>  
        </div>  
        <div class="col mx-2 bg-primary text-white">  
            <p><strong>ISBN:</strong> {{ book.isbn }}</p>  
        </div>  
        <div class="col mx-2 bg-primary text-white">  
            <p><strong>Язык:</strong> {{ book.language }}</p>  
        </div>  
        <div class="col mx-2 bg-primary text-white">  
            <p><strong>Издательство:</strong> {{book.publisher}}</p>  
        </div>  
    </div>  
    <div style="margin-left:20px; margin-top:10px">  
        <h4>Количество экземпляров книг в БД</h4>  
        {% for copy in book.bookinstance_set.all %}  
            <hr>  
            <p class="text-muted">  
                <strong>Инвентарный номер:</strong> {{copy.id}}</p>  
            <p class="text-success">  
                <strong>Статус экземпляра книги:</strong> {{copy.status}}</p>  
            {% endfor %}  
    </div>
```

```
</div>
{% endblock %}
```

Здесь, как и в шаблоне главной страницы, в первой строке мы расширяем наш базовый шаблон — `{% extends "catalog/base.html" %}`, а затем определяем блок с именем `content`. Внутри этого блока выдаются данные о конкретной книге.

Сначала выводятся две строки: название книги, автор (авторы). Так как авторов может быть несколько, то здесь с использованием инструкции `if` организован цикл перебора всех авторов, а в теле цикла выводятся имя автора (`author.first_name`), фамилия автора (`author.last_name`).

После этого выводится изображение обложки с ограничением по высоте 200 px, а рядом с ней кнопка со ссылкой на URL-адрес изображения обложки. Нажатие на эту кнопку обеспечивает вывод обложки в полном размере в отдельной вкладке браузера. Под обложкой выводится цена книги (`book.price`). После этого следует строка для вывода аннотации к книге (`book.summary`).

Затем с использованием тегов Bootstrap создано четыре адаптивных блока, в которых выводятся:

- жанр книги (`book.genre`);
- ISBN книги (`book.isbn`);
- язык, на котором написана книга (`book.language`);
- издательство (`book.publisher`).

В заключительном блоке с инструкцией `if` организован цикл, в котором выводятся сведения о состоянии конкретных экземпляров данной книги. Здесь интересным методом, который мы не встречали ранее, является метод `book.bookinstance_set.all`. Этот метод Django формирует автоматически. Он обеспечивает получение множества записей о состоянии экземпляров (`BookInstance`) конкретной книги (`Book`):

```
{% for copy in book.bookinstance_set.all %}
    <!-- итерации по каждому экземпляру книги -->
{% endfor %}
```

Этот метод создан потому, что в модели данных была указана связь «один ко многим» (одна книга → много экземпляров книг). С учетом этой связи Django самостоятельно конструирует метод «обратного просмотра» (`reverse lookup`), которым можно воспользоваться. Имя этого метода создается автоматически с использованием символов в нижнем регистре и состоит из имени модели, в которой был объявлен первичный ключ — `ForeignKey` (в нашем случае это `bookinstance`), за которым следует выражение `_set`. В нашем примере метод, созданный для модели `Book`, имеет имя `book.bookinstance_set`. В приведенном шаблоне также был указан параметр `all` для получения всех записей из таблицы БД (`book.bookinstance_set.all`).

К настоящему моменту мы создали все необходимое для показа полной информации о каждой книге. Запускаем наш сервер (`python manage.py runserver`) и открываем браузер: <http://127.0.0.1:8000/>. На главной странице в меню выбираем опцию **Все книги** и получаем страницу со списком книг. Если теперь щелкнуть левой кнопкой мыши на назва-

нии книги, например **Война и мир**, то будет выдана страница с полной информацией об этой книге (рис. 10.9).

Обратите внимание, что на данной странице фамилия и имя автора имеют синий цвет, а это признак того, что они являются ссылкой для перехода на страницу сайта с информацией об авторах. Пока эта ссылка не работает, т. к. мы еще не создали соответствующую страницу и не реализовали механизм доступа к ней. Это мы сделаем несколько позже.

Мир книг - печатные и электронные интерактивные книги

Главная страница О компании Контакты

Все книги  
Все авторы

## Название книги - "Война и мир"

Автор: [Лев Толстой](#)

 Показать

Цена: 2500,00 руб.

Аннотация: "Война и мир" Л. Н. Толстого - книга на все времена. Как будто она существовала всегда, настолько знакомым кажется текст, едва мы отираем первые страницы романа, настолько памятны многие его эпизоды: охота и свадьбы, первый бал Наташи Ростовой, лунная ночь в Отрадном, князь Андрей в сражении при Аустерлице... Сцены "мирной", семейной жизни сменяются картинами, имеющими значение для хода всей мировой истории. Но для Толстого они равнозначны, связанны в едином потоке времени. Каждый эпизод важен не только для развития сюжета, но и как одно из бесчисленных проявлений жизни, которую учит любить Толстой.

Жанр: Проза ISBN: 978-5-389-062 Язык: Русский Издательство: ЛитРес

### Количество экземпляров книг в БД

Инвентарный номер: 6  
Статус экземпляра книги: На складе

Инвентарный номер: 7  
Статус экземпляра книги: Продана

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены.

Рис. 10.9. Страница с полной информацией о книге

А теперь выдадим сведения о книге, у которой несколько авторов, например «12 стульев». В этом случае в шаблоне страницы с использованием цикла будет выдана информация о двух авторах книги (рис. 10.10).



Рис. 10.10. Страница с детальной информацией о книге двух авторов

## 10.6. Постраничный вывод большого числа записей из БД (класс *Paginator*)

Если в БД всего несколько записей об объекте, то наша страница вывода списка книг будет выглядеть отлично. Тем не менее, когда в БД появятся десятки или сотни записей, страница станет значительно дольше загружаться и окажется слишком длинной для комфорtnого просмотра. Решением этой проблемы может стать добавление постраничного вывода (*Pagination*) к отображению списка книг. При этом на каждую страницу будет выводиться ограниченное число элементов.

Django имеет отличный встроенный механизм для организации постраничного вывода. Более того, он встроен в класс *ListView* отображения списков, так что нам не придется проделывать большой объем работы, чтобы воспользоваться возможностями постраничного вывода.

Откроем файл *catalog\views.py* и добавим объект *paginate\_by* (листинг 10.16, изменения выделены серым фоном).

### Листинг 10.16. Измененный код файла *catalog\views.py*

```
class BookListView(ListView):
    model = Book
    context_object_name = 'books'
    paginate_by = 3
```

Как только в базе данных появится более трех записей, представление начнет формировать постраничный вывод данных (страницы будут передаваться шаблону). К раз-

личным страницам можно будет получить доступ при помощи параметров GET-запроса. Например, к странице 2 можно получить доступ через URL-адрес `/catalog/books/?page=2`.

Теперь, когда задан вывод данных постранично, нам нужно добавить функционал переключения между страницами в шаблон страницы. Поскольку целесообразно использовать этот механизм для всех списков на сайте, то мы пропишем его в базовом шаблоне сайта.

Откроем файл `\WebBooks\templates\catalog\base.html` и после блока `content` вставим блок, отвечающий за постраничный вывод (листинг 10.17, изменения выделены серым фоном).

#### Листинг 10.17. Измененный код файла `\WebBooks\templates\catalog\base.html`

```
<!DOCTYPE html>
<html lang="en">
<head>
    {%- load static %}
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="/static/css/bootstrap.min.css" >
    <script defer src="/static/js/bootstrap.bundle.min.js"></script>
</head>
<body>
    <div class="container-fluid p-1 bg-primary text-white text-center">
        <div class="row">
            <div class="col-2 text-start">
                
            </div>
            <div class="col-10 ">
                <h3>Мир книг – печатные и электронные интерактивные книги</h3>
            </div>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row bg-warning text-center">
            <h6>
                <a href="{% url 'index' %}">Главная страница</a>
                <a href="{% url 'index' %}">О компании</a>
                <a href="{% url 'index' %}">Контакты</a>
            </h6>
        </div>
    </div>
    <div class="container-fluid">
        <div class="row text-center text-primary fs-1 fw-bold">
            <div>{%- block header%}
                {%- endblock header %}
            </div>
        </div>
        <div class="row text-start text-body">
            <div class="col-2 ">
```

```
{% block sidebar %}  
<nav class="nav flex-column">  
    <a class="nav-link" href="{% url 'books-list' %}">Все книги</a>  
    <a class="nav-link" href="{% url 'authors-list' %}">Все авторы</a>  
</nav>  
{% endblock sidebar %}  
</div>  
<div class="col-10" >  
    <div class="row text-center">  
        {% block content%}  
        {% endblock content %}  
  
        <!-- пагинатор -->  
        <div class="pagination">  
            <span class="step-links">  
                {% if page_obj.paginator.num_pages > 1 %}  
                    {% if page_obj.has_previous %}  
                        <a href="?page=1">&laquo; Первая</a>  
                        <a href="?page={{ page_obj.previous_page_number }}">Предыдущая</a>  
                    {% endif %}  
  
                    <span class="current">  
                        Стр. {{ page_obj.number }}  
                        из {{ page_obj.paginator.num_pages }}.  
                    </span>  
  
                    {% if page_obj.has_next %}  
                        <a href="?page={{ page_obj.next_page_number }}">Следующая</a>  
                        <a href="?page={{ page_obj.paginator.num_pages }}">Предыдущая &raquo;</a>  
                    {% endif %}  
                {% endif %}  
            </span>  
        </div>  
        <!-- пагинатор -->  
    </div>  
</div>  
<div class="container-fluid">  
    <div class="row bg-info text-center text-dark small lh-sm pt-2 my-2">  
        {% block footer %}  
            <p>Copyright ООО "Интеллектуальные информационные системы",  
            2023. Все права защищены</p>  
        {% endblock footer%}  
    </div>  
</div>  
</body>  
</html>
```

Чтобы более детально изучить структуру кода класса pagination, выделим его из базового шаблона и покажем с учетом правильных отступов:

```
<!-- пагинатор -->
<div class="pagination">
    <span class="step-links">
        {# if page_obj.paginator.num_pages > 1 #}
        {# if page_obj.has_previous #}
            <a href="?page=1">&laquo; Первая</a>
            <a href="?page={{ page_obj.previous_page_number }}">Предыдущая</a>
        {# endif #}
        <span class="current">
            Стр. {{ page_obj.number }} из {{ page_obj.paginator.num_pages }}.
        </span>
        {# if page_obj.has_next #}
            <a href="?page={{ page_obj.next_page_number }}">Следующая</a>
            <a href="?page={{ page_obj.paginator.num_pages }}">Предыдущая &raquo;</a>
        {# endif #}
        {# endif #}
    </span>
</div>
<!-- пагинатор -->
```

В этом коде сначала выполняется проверка, что число выводимых страниц больше одной (`if page_obj.paginator.num_pages > 1`). Если число выводимых страниц меньше или равно единице, то выводимый список нет смысла разбивать на страницы. Если же список не помещается на одной странице, то далее делается проверка, включен ли механизм постраничного вывода. Если разбивка списка на страницы включена, то после вывода первой страницы списка отображаются следующие элементы:

- номер текущей страницы (`page_obj.number`);
- общее число страниц (`page_obj.paginator.num_pages`);
- надписи со ссылками на страницы **Следующая** и **Предыдущая**.

Здесь параметр `page_obj` является объектом типа `Paginator`, который станет создаваться каждый раз, когда будет применяться постраничный вывод данных для текущей страницы. Он позволяет получить всю информацию о текущей странице, о предыдущих страницах, об общем числе страниц и т. п.

В теле класса `pagination` есть два блока с инструкцией `if`:

- `{% if page_obj.has_previous %}` — проверка, поступил ли запрос на переход к предыдущей странице;
- `{% if page_obj.has_next %}` — проверка, поступил ли запрос на переход к следующей странице.

Если условие проверки соблюдается, то выполняется переход к заданной странице. Здесь используем выражение `<href=?page=>` для получения URL-адреса нужной страницы.

Если мы теперь загрузим страницу со списком книг, то под ним увидим блок организации постраничного вывода данных (рис. 10.11).

Нажмите в нем на ссылку **Следующая** — будет выведена следующая страница со списком книг и появится ссылка для возврата на предыдущую страницу (рис. 10.12).

**Мир книг - печатные и электронные интерактивные книги**

Главная страница О компании Контакты

Список книг в БД

Название	Автор	Жанр	Обложка	Показать обложку
<a href="#">12 стульев</a>	Ильф, Петров	Приключения		<a href="#">Показать</a>
<a href="#">Золотой теленок</a>	Ильф, Петров	Приключения		<a href="#">Показать</a>
<a href="#">Война и мир</a>	Толстой	Проза		<a href="#">Показать</a>

Стр. 1 из 2. Следующая Предыдущая »

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены!

A callout bubble points from the text "Постстраничный вывод списка книг" in the third row of the table to the "Показать" button in the same row.

Рис. 10.11. Блок организации постраничного вывода данных

**Мир книг - печатные и электронные интерактивные книги**

Главная страница О компании Контакты

Список книг в БД

Название	Автор	Жанр	Обложка	Показать обложку
<a href="#">Дубровский</a>	Пушкин	Проза		<a href="#">Показать</a>
<a href="#">Продавец воздуха</a>	Беляев	Фантастика		<a href="#">Показать</a>

« Первая Предыдущая Стр. 2 из 2.

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены!

A callout bubble points from the text "Возврат к предыдущим страницам" in the second row of the table to the "Показать" button in the same row.

Рис. 10.12. Листание списка страниц с данными с возможностью возврата на предыдущие страницы

## 10.7. Создание страницы со списком авторов на основе класса *ListView*

В предыдущих разделах в шаблонах были предусмотрены ссылки для получения информации об авторах книг, но они пока не работают, т. к. не были созданы соответствующие страницы и не прописаны маршруты к этим страницам. Реализуем механизм показа списка авторов. Страница со списком авторов будет доступна по адресу **catalog/authors/**. Начнем мы с сопоставления URL-адресов, для чего откроем файл `\WebBooks\WebBooks\catalog\urls.py` и добавим в него строку, которая выделена в листинге 10.18 серым фоном.

### Листинг 10.18. Измененный код файла `\WebBooks\WebBooks\catalog\urls.py`

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
]
```

Здесь функция `path()` связывает маршрут (URL-адрес) страницы (`authors/`) с классом в представлении `views.AuthorListView.as_view()`. Кроме того, здесь определено имя для ссылки на данную страницу сайта (`name='authors_list'`).

Откроем файл `catalog\views.py` и добавим в него класс `AuthorsListView` с помощью кода листинга 10.19 (изменения выделены серым фоном).

### Листинг 10.19. Измененный код файла `catalog\views.py`

```
from .models import Book, Author, BookInstance
class AuthorListView(ListView):
    model = Author
    paginate_by = 4
```

В созданном классе по умолчанию будет сформирован ряд объектов, в частности:

- `author_list` — список с данными об авторах, полученных из БД;
- `template_name` — имя шаблона для вывода списка авторов (по умолчанию `authors_list.html`).

На следующем шаге создадим в каталоге `\WebBooks\templates\catalog\` новый HTML-файл `author_list.html` (шаблон для выдачи списка авторов книг) и напишем в нем код листинга 10.20.

### Листинг 10.20. Измененный код файла `\WebBooks\templates\catalog\author_list.html`

```
{% extends "catalog/base.html" %}
{% block content %}
```

```

<h4>Список авторов книг</h4>
{% if author_list %}
    <table class="table table-striped table-bordered text-start">
        <thead>
            <tr>
                <th>Автор</th>
                <th>Фото</th>
                <th>Показать фото</th>
            </tr>
        </thead>
        <tbody>
            {% for author in author_list.all %}
            <tr>
                <td><a href="{{ author.id }}>{{ author.first_name }}<br>{{ author.last_name }} </a></td>
                <td></td>
                <td><a href="{{ author.photo.url }}" class="btn btn-primary" target="_blank"> Показать</a></td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
{% else %}
    <p>В базе данных нет авторов</p>
{% endif %}
{% endblock %}

```

Здесь, как и в шаблонах других страниц страницы, в первой строке мы расширяем наш базовый шаблон — `{% extends "catalog/base.html" %}`, а затем определяем блок с именем `content`. Внутри блока в цикле формируется и выдается список авторов книг. В этом шаблоне для доступа к данным об авторах используется объект `author_list`.

Сначала с помощью инструкции `if` проверяется, есть ли в БД записи с информацией об авторах — `{% if author_list %}`. Если в БД есть хотя бы одна запись об авторе книги, то создается таблица Bootstrap, если в БД нет записей об авторах, то пользователю будет выдано сообщение «В базе данных нет авторов».

В теге Bootstrap `<table>` создано три колонки. Сначала в теге `<thead>` формируется «шапка» таблицы, а затем в цикле `{% for author in author_list.all %}` столбцы таблицы заполняются следующей информацией:

- `{{ author.first_name }}` — имя автора;
  - `{{ author.last_name }}` — фамилия автора;
  - `{{ author.photo.url }}` — фотография (портрет) автора (с размером по высоте не более 100 px);
  - `href="{{ author.photo.url }}"` — кнопка для показа полноразмерного портрета автора;
- Здесь текст с именем и фамилией автора представляет собой ссылку, которая дает возможность перейти на страницу с полной информацией об этом авторе. В качестве параметра в URL-адрес передается `author.id`.

Теперь нужно открыть файл с базовым шаблоном по пути

**\WebBooks\templates\catalog\base.html**

и указать URL-ссылку для пункта бокового меню **Все авторы**, как показано в листинге 10.21 (изменения выделены серым фоном).

**Листинг 10.21. Измененный код файла \WebBooks\templates\catalog\base.html**

```
{% block sidebar %}  
<nav class="nav flex-column">  
    <a class="nav-link" href="{% url 'books-list' %}">Все книги</a>  
    <a class="nav-link" href="{% url 'authors-list' %}">Все авторы</a>  
</nav>  
{% endblock sidebar %}
```

Здесь мы создали переход на страницу с авторами книг (теперь мы можем смело это сделать, поскольку у нас имеется соответствующее URL-преобразование). Для выдачи списка авторов у нас все сделано. Загружаем главную страницу нашего сайта и в левом меню щелкаем левой кнопкой мыши на ссылке **Все авторы** — откроется страница с полным списком авторов (рис. 10.13).

Список авторов книг		
	Автор	Фото
Показать		
	<a href="#">Александр Беляев</a>	
	<a href="#">Александр Пушкин</a>	
	<a href="#">Лев Толстой</a>	
	<a href="#">Илья Ильф</a>	

Стр. 1 из 2. Следующая Предыдущая »

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены.

**Рис. 10.13. Страница сайта с отображением списка авторов книг**

Следует обратить внимание, что список авторов разбит на страницы. Это автоматически сработал блок разбивки данных на страницы, который мы ранее создали в базовом шаблоне.

## 10.8. Создание страницы с детальной информацией об авторе книги на основе класса *DetailView*

Доступ к странице с детальной информацией об авторе книги осуществляется при помощи URL-адреса `catalog/author/<id>` (`<id>` — первичный ключ в БД для этого автора). В детальную информацию об авторе в дополнение к полям модели `Author`, которые отображены в списке авторов (имя, фамилия и фото), мы добавим день рождения и краткие сведения об авторе.

Откроем файл `WebBooks\catalog\urls.py` и добавим строку в URL-преобразование, как показано в листинге 10.22 (изменения выделены серым фоном).

### Листинг 10.22. Измененный код файла `WebBooks\catalog\urls.py`

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),
         name='authors-detail'),
]
```

Практически так же, как и для страницы сайта `authors`, функция `path()` связывает маршрут (URL-адрес) страницы (`authors/<int:pk>/`) с классом в представлении `views.AuthorDetailView.as_view()`. Кроме того, здесь определено имя для ссылки на данную страницу сайта (`name='authors-detail'`).

Поскольку мы заранее не знаем, о каком авторе будет запрошена информация, то в этом случае потребуется функция `path` с параметрами формирования URL-адреса (`authors/<int:pk>/`). Здесь угловые скобки `<..>` необходимы для того, чтобы получить значение из URL-адреса. Преобразователь `int` обеспечивает гарантированное получение целочисленного параметра, а параметр `pk` (primary key — первичный ключ) служит для получения идентификатора автора из БД.

Здесь мы воспользуемся классом, который наследуется от существующего в Django класса `DetailView` (отображение элемента из списка данных). В классе `DetailView` уже реализован функционал для того, чтобы получить и показать информацию о конкретном авторе, а значит, мы справимся со своей задачей меньшим объемом программного кода.

Откроем файл `catalog\views.py` и добавим в него класс `AuthorDetailView` с помощью кода листинга 10.23.

**Листинг 10.23. Измененный код файла catalog\views.py**

```
class AuthorDetailView(DetailView):
    model = Author
```

Вот, собственно, и весь код, который нужно добавить в данный модуль.

Здесь мы на основе базового класса (`DetailView`) создали собственный класс `AuthorDetailView(DetailView)`. В созданном классе по умолчанию будет сформирован ряд объектов, в частности:

- `object` — данные о конкретном авторе, полученные из БД;
- `template_name` — имя шаблона для вывода информации о конкретном авторе (по умолчанию `author_detail.html`).

Мы оставили имя шаблона для вывода информации о книге без изменения (`author_detail.html`). В итоге созданный нами класс `DetailView` выполнит запрос к базе данных, получит данные из БД о конкретном авторе, а затем передаст эти данные в шаблон `author_detail.html`. В этом шаблоне теперь из объекта `object` (или `author`) можно получить данные о конкретном авторе: `object.first_name` — имя автора, `object.last_name` — фамилия автора, `object.year` — год рождения и т. д.

**ПРИМЕЧАНИЕ**

Ту же самую информацию можно получить и из объекта `author`, например: `author.first_name` — имя автора, `author.last_name` — фамилия автора, `author.year` — год рождения.

Теперь нужно создать шаблон для выдачи пользователю данных об авторе с именем `author_detail.html` и поместить его по следующему пути:

`\WebBooks\templates\catalog\author_detail.html`.

В этом шаблоне напишем код листинга 10.24.

**Листинг 10.24. Измененный код файла \WebBooks\templates\catalog\author\_detail.html**

```
% extends "catalog/base.html"
% block content %
<div class="container text-start">
    <div class="row my-2">
        <h4>{{ object.first_name }} {{author.last_name}}</h4>
        <p>Родился - {{author.date_of_birth}}</p>
        <p>Фото (портрет)</p>
        <p></p>
    <div class="row my-2">
        <div class="col-2 my-2">
            <a href="{{ author.photo.url }}" class="btn btn-primary"
               target="_blank"> Показать </a>
        </div>
    </div>
    <div class="row my-2">
        {{author.about}}
    </div>
```

```
</div>
</div>
{% endblock %}
```

Здесь, как и в шаблонах других страниц, в первой строке мы расширяем наш базовый шаблон — `{% extends "catalog/base.html" %}`, а затем определяем блок с именем `content`. Внутри этого блока выдаются данные о конкретном авторе:

- `object.first_name` — имя;
- `author.last_name` — фамилия;
- `author.date_of_birth` — дата рождения;
- `author.photo.url` — портрет;
- `author.about` — краткая характеристика автора.

Обратите внимание, что в этом модуле сведения об авторе были получены из объекта `object` и из объекта `author`.

К настоящему моменту мы создали все необходимое для показа полной информации о каждом авторе. Запускаем наш сервер (`python manage.py runserver`) и открываем браузер: `http://127.0.0.1:8000/`. На главной странице в меню выбираем опцию **Все авторы** и получаем страницу со списком авторов. Если теперь щелкнуть левой кнопкой мыши на имени или фамилии автора, например **Лев Толстой**, то будет выдана страница с полной информацией об этом авторе (рис. 10.14).



Рис. 10.14. Страница со сведениями об авторе

Как видно из данного рисунка, на HTML-странице не отображается пагинатор, т. к. в этом случае страница всегда будет состоять из одного листа.

У нас осталась еще одна неработающая ссылка из шаблона book\_detail.html — показать авторов книги (ссылка выделена серым фоном):

```
{% for author in book.author.all %}
    <a href="#"> {{ author.first\_name }} {{author.last\_name}}</a>
{% endfor %}
```

Мы оставили эту ссылку пустой, т. к. на тот момент у нас еще не было создано страницы для показа информации об авторе. Поскольку теперь такая страница существует, в этом месте можно написать код вызова страницы с детальной информацией об авторе. Открываем шаблон book\_detail.html и корректируем одну строку (листинг 10.25, изменения выделены серым фоном).

#### Листинг 10.25. Измененный код файла \WebBooks\templates\catalog\book\_detail.html

```
<div class="row my-2">
    <p><strong>Автор:</strong>
    {% for author in book.author.all %}
        <a href="/authors/{{ author.id }}">
            {{ author.first_name }} {{author.last_name}}</a>
    {% endfor %}
</div>
```

В этой строке присутствует ссылка на URL `href="/authors/{{author.id}}"`, т. е. на страницу /authors/, куда передается идентификатор автора в БД — `{{author.id}}`.

Теперь если открыть страницу с детальной информацией о любой книге, то ссылки на имя или фамилию автора будут рабочими (рис. 10.15).

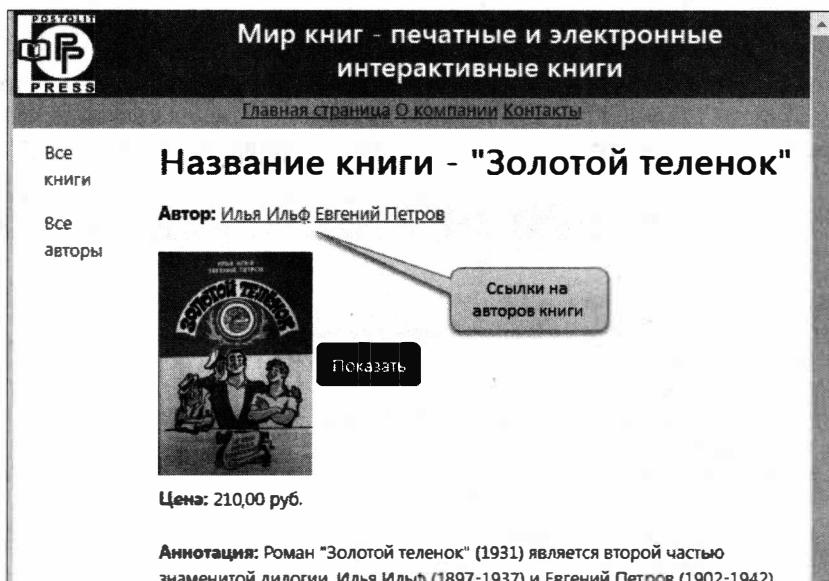


Рис. 10.15. Рабочие ссылки на авторов книг

Итак, мы закончили формирование всех страниц для отображения информации о книгах и об авторах. До полного комплекта на этом сайте не хватает еще двух страниц, на которые есть ссылки из главного меню: **О компании** и **Контакты**. Создадим эти недостающие страницы.

## 10.9. Создание страниц О компании и Контакты

Создание этих двух страниц не представляет особых трудностей, т. к. они в нашем примере будут статичными. Нам нужно создать два URL-адреса: `catalog/about/` и `catalog/contact/`, а также два шаблона страниц: `about.html` и `contact.html`.

Откроем файл `\WebBooks\catalog\urls.py` и добавим две строки в URL-преобразование, как показано в листинге 10.26 (изменения выделены серым фоном).

### Листинг 10.26. Измененный код файла `\WebBooks\catalog\urls.py`

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),
         name='authors-detail'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
]
```

Здесь добавлены маршруты к страницам `about` (О компании) и `contact` (Контакты).

Теперь откроем файл `catalog\views.py` и добавим в него две функции: `about` для взаимодействия со страницей О компании, и `contact` для взаимодействия со страницей Контакты. Код, который нужно добавить в файл `views.py`, приведен в листинге 10.27.

### Листинг 10.27. Измененный код файла `\WebBooks\catalog\views.py`

```
def about(request):
    text_head = 'Сведения о компании'
    name = 'ООО "Интеллектуальные информационные системы"'
    rab1 = 'Разработка приложений на основе' \
           ' систем искусственного интеллекта'
    rab2 = 'Распознавание объектов дорожной инфраструктуры'
    rab3 = 'Создание графических АРТ-объектов на основе' \
           ' систем искусственного интеллекта'
    rab4 = 'Создание цифровых интерактивных книг, учебных пособий' \
           ' автоматизированных обучающих систем'
    context = {'text_head': text_head, 'name': name,
               'rab1': rab1, 'rab2': rab2,
               'rab3': rab3, 'rab4': rab4}
```

```
# передача словаря context с данными в шаблон
return render(request, 'catalog/about.html', context)

def contact(request):
    text_head = 'Контакты'
    name = 'ООО "Интеллектуальные информационные системы"'
    address = 'Москва, ул. Планерная, д. 20, к. 1'
    tel = '495-345-45-45'
    email = 'iis_info@mail.ru'
    # Словарь для передачи данных в шаблон index.html
    context = {'text_head': text_head,
               'name': name, 'address': address,
               'tel': tel,
               'email': email}
    # передача словаря context с данными в шаблон
    return render(request, 'catalog/contact.html', context)
```

Это достаточно простой и уже знакомый нам код. Здесь нет обращений к БД, просто сформировано несколько текстовых переменных, все их значения упакованы в словарь context, который передается в шаблоны: about.html и contact.html. Теперь необходимо создать эти шаблоны.

Для вывода сведений о компании создадим шаблон с именем about.html и поместим его по следующему пути:

**\WebBooks\templates\catalog\about.html.**

В этом шаблоне напишем код листинга 10.28.

#### Листинг 10.28. Код файла \WebBooks\templates\catalog\about.html

```
{% extends "catalog/base.html" %}

<head>
{% load static %}
</head>

{% block content %}


![image]({% static 'images/ai.jpg' %})

## Искусственный интеллект



### {%rab1%}



![image]({% static 'images/road.jpg' %})

## Мир книг



### {%rab2%}


```

```
<h2 class="text-white">Компьютерное зрение</h2>
<h3 class="text-white">{{rab2}}</h3>
</div>
</div>
<div class="carousel-item">
    
    <div class="carousel-caption d-none d-md-block">
        <h2 class="text-white">Арт-объекты</h2>
        <h3 class="text-white">{{rab3}}</h3>
    </div>
</div>
<div class="carousel-item">
    
    <div class="carousel-caption d-none d-md-block">
        <h2 class="text-white">Электронные книги</h2>
        <h3 class="text-white">{{rab4}}</h3>
    </div>
</div>
</div>
<button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleControls" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Предыдущий</span>
</button>
<button class="carousel-control-next" type="button" data-bs-target="#carouselExampleControls" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Следующий</span>
</button>
</div>
</div>
<!-- Конец карусели --&gt;
&lt;div class="container-fluid"&gt;
    &lt;h4 class="text-center"&gt;{{ text_head }}&lt;/h4&gt;
    &lt;div class="row my-2 text-center"&gt;
        &lt;div class="col-5"&gt;&lt;/div&gt;
        &lt;div class="col-1 text-center"&gt;
            &lt;img src="{% static 'images/SIS.jpg' %}"&gt;
        &lt;/div&gt;
        &lt;div class="col-5"&gt;&lt;/div&gt;
    &lt;/div&gt;
    &lt;div class="row my-2 text-primary text-center"&gt;
        &lt;h3&gt;{{ name }}&lt;/h3&gt;
    &lt;/div&gt;
    &lt;div class="row my-2 text-white text-start"&gt;
        &lt;div class="col mx-2 my-2 bg-primary"&gt;{{rab1}}&lt;/div&gt;
        &lt;div class="col mx-2 my-2 bg-primary"&gt;{{rab2}}&lt;/div&gt;
        &lt;div class="col mx-2 my-2 bg-primary"&gt;{{rab3}}&lt;/div&gt;
        &lt;div class="col mx-2 my-2 bg-primary"&gt;{{rab4}}&lt;/div&gt;
    &lt;/div&gt;
&lt;/div&gt;</pre>
```

```
<div class="row my-2 text-center">
    <!-- пустые колонки слева от аккордеона--&gt;
    &lt;div class="col-3"&gt;&lt;/div&gt;
    <!-- 6 колонок с аккордеоном --&gt;
    &lt;div class="col-6 text-start"&gt;
        <!-- Аккордеон --&gt;
        &lt;div class="accordion" id="accordionExample"&gt;
            <!-- Аккордеон элемент 1--&gt;
            &lt;div class="accordion-item"&gt;
                &lt;h2 class="accordion-header" id="headingOne"&gt;
                    &lt;button class="accordion-button" type="button"
                        data-bs-toggle="collapse"
                        data-bs-target="#collapseOne"
                        aria-expanded="true" aria-controls="collapseOne"&gt;
                        Искусственный интеллект
                    &lt;/button&gt;
                &lt;/h2&gt;
                &lt;div id="collapseOne" class="accordion-collapse collapse"
                    aria-labelledby="headingOne" data-bs-parent="#accordionExample"&gt;
                    &lt;div class="accordion-body"&gt;
                        &lt;strong&gt;Разработка под ключ.&lt;/strong&gt; Разрабатываем
                        информационные комплексы на базе систем
                        искусственного интеллекта.
                    &lt;/div&gt;
                &lt;/div&gt;
            &lt;/div&gt;
            <!-- Аккордеон элемент 2--&gt;
            &lt;div class="accordion-item"&gt;
                &lt;h2 class="accordion-header" id="headingTwo"&gt;
                    &lt;button class="accordion-button collapsed" type="button"
                        data-bs-toggle="collapse" data-bs-target="#collapseTwo"
                        aria-expanded="false" aria-controls="collapseTwo"&gt;
                        Дорожная инфраструктура
                    &lt;/button&gt;
                &lt;/h2&gt;
                &lt;div id="collapseTwo" class="accordion-collapse collapse"
                    aria-labelledby="headingTwo" data-bs-parent="#accordionExample"&gt;
                    &lt;div class="accordion-body"&gt;
                        &lt;strong&gt;Системы распознавания.&lt;/strong&gt; Разрабатываем системы
                        распознавания объектов дорожной инфраструктуры с использованием
                        нейронных сетей.
                    &lt;/div&gt;
                &lt;/div&gt;
            &lt;/div&gt;
            <!-- Аккордеон элемент 3--&gt;
            &lt;div class="accordion-item"&gt;
                &lt;h2 class="accordion-header" id="headingThree"&gt;
                    &lt;button class="accordion-button collapsed" type="button"
                        data-bs-toggle="collapse"
                        data-bs-target="#collapseThree"
                        aria-expanded="false" aria-controls="collapseThree"&gt;</pre>
```

```
Арт-объекты
</button>
</h2>
<div id="collapseThree" class="accordion-collapse collapse"
      aria-labelledby="headingThree" data-bs-parent="#accordionExample">
<div class="accordion-body">
    <strong>Обработка изображений.</strong> Создаем уникальные изображения
        с использованием систем машинного зрения и искусственного интеллекта.
</div>
</div>
<!-- Аккордеон элемент 4-->
<div class="accordion-item">
    <h2 class="accordion-header" id="headingFour">
        <button class="accordion-button collapsed" type="button"
               data-bs-toggle="collapse"
               data-bs-target="#collapseFour"
               aria-expanded="false" aria-controls="collapseThree">
            Интерактивные электронные книги
        </button>
    </h2>
    <div id="collapseFour" class="accordion-collapse collapse"
          aria-labelledby="headingFour" data-bs-parent="#accordionExample">
        <div class="accordion-body">
            <strong>Интерактивное обучение</strong> Разрабатываем
                интерактивные обучающие пособия, учебники, электронные
                книги для студентов и школьников.
        </div>
    </div>
</div>
<!-- Аккордеон конец-->
</div>
</div>
<!-- пустые колонки справа от аккордеона-->
<div class="col-3"></div>
</div>
{ % endblock %}
```

Листинг данного шаблона достаточно длинный, но не стоит этого пугаться. Здесь показан пример использования на HTML-страницах двух новых элементов Bootstrap: «карусель» (carousel) и «аккордеон» (accordion). Именно эти два элемента занимают большую часть программного кода. Это достаточно популярные элементы интерфейса, которые используются на страницах современных сайтов. «Карусель» позволяет создать область автоматической прокрутки слайдов в «шапке» страницы, а «аккордеон» — группу раскрывающихся списков. Блоки программного кода с этими элементами выделены строками с соответствующими комментариями.

Для вывода сведений с контактными данными компании создадим шаблон с именем contact.html и поместим его по следующему пути:

**\WebBooks\templates\catalog\contact.html.**

В этом шаблоне напишем код листинга 10.29.

**Листинг 10.29. Код файла WebBooks\templates\catalog\contact.html**

```
{% extends "catalog/base.html" %}

<head>
{% load static %}
</head>

{% block content %}

<div class="container-fluid">
    <div class="row my-2 text-center">
        <div class="col-5"></div>
        <div class="col-1 text-center">
            
        </div>
        <div class="col-5"></div>
    </div>
    <div class="row my-2 text-primary text-center">
        <h3>{{ name }}</h3>
    </div>
    <div class="row my-2 text-muted ">
        <h4 class="text-center">{{ text_head }}</h4>
    </div>
    <div class="row my-2">
        <!-- пустые колонки слева от таблицы-->
        <div class="col-3"></div>
        <div class="col-6">
            <table class="table table-striped table-bordered text-start">
                <thead>
                    <tr>
                        <th>Адрес</th>
                        <th>Телефон</th>
                        <th>e-mail</th>
                    </tr>
                </thead>
                <tbody>
                    <tr>
                        <td>{{ address }}</td>
                        <td>{{ tel }}</td>
                        <td>{{ email }}</td>
                    </tr>
                </tbody>
            </table>
        </div>
        <!-- пустые колонки справа от таблицы-->
        <div class="col-3"></div>
        <!-- карта-->
        <div class="row my-2 text-center">
            <div class="col-2"></div>
            <div class="col-8">
                
            </div>
        </div>
    </div>
</div>
```

```

<div class="col-2"></div>
</div>
<% endblock %>

```

Код этого шаблона значительно проще, т. к. используются уже знакомые традиционные элементы. Здесь выводится таблица с контактными данными и изображение карты с указанием места расположения офиса компании.

Остался последний шаг — нужно в главном меню активировать ссылки на страницы О компании и Контакты. В базовом шаблоне `base.html` изменим две строчки (листинг 10.30, изменения выделены серым фоном).



Рис. 10.16. Страница сайта О компании

**Листинг 10.30. Измененный код файла \WebBooks\templates\catalog\base.html**

```
<div class="container-fluid">
    <div class="row bg-warning text-center">
        <h6>
            <a href="{% url 'index' %}">Главная страница</a>
            <a href="{% url 'about' %}">О компании</a>
            <a href="{% url 'contact' %}">Контакты</a>
        </h6>
    </div>
</div>
```

К настоящему моменту мы создали все страницы сайта, которые были запланированы на начальном этапе проектирования. Запускаем наш сервер (`python manage.py runserver`) и открываем браузер: `http://127.0.0.1:8000/`. На главной странице в меню выбираем опцию **О компании**, после этого будет выдана страница с информацией о деятельности компании (рис. 10.16).

В верхней части страницы мы видим элемент «карусель», обеспечивающий прокрутку изображений, которые были загружены в папку `static/images`. В нижней части страницы находится элемент «аккордеон», это, по сути, раскрывающийся список. При нажатии на стрелку в правой части элемента раскроется блок, в котором будет показана текстовая информация (рис. 10.17).

Ну вот мы и закончили разработку базовых страниц нашего учебного сайта, теперь можно перейти к изучению материалов следующей главы.

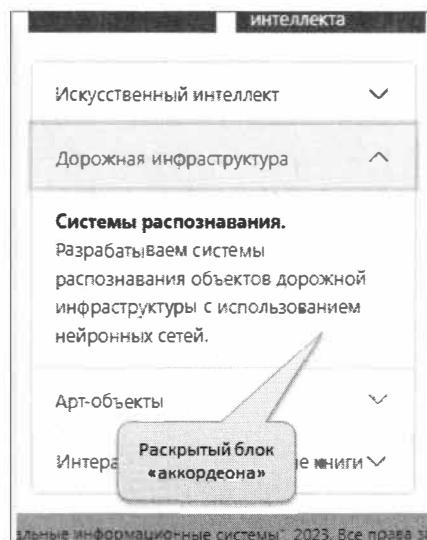


Рис. 10.17. Пример раскрывшегося блока элемента «аккордеон»

## 10.10. Краткие итоги

В этой главе мы рассмотрели применение обобщенных классов для отображения списка данных и разобрались с показом детальной информации об отдельных элементах списка (на примере показа списка названий книг и детальной информации о книге). Кроме того, мы многое узнали о шаблонах преобразования URL-адресов, построенных на основе регулярных выражений, а также о том, как можно передавать данные из URL-адреса в представление (view). Мы изучили несколько приемов применения шаблонов и в самом конце главы показали, как осуществлять постраничный вывод длинных списков.

В следующей главе мы расширим наш сайт «Мир книг», введя в него поддержку пользовательских аккаунтов, и таким образом продемонстрируем аутентификацию, разграничение уровней доступа и познакомимся с понятием *сессии*. А также разработаем пользовательские формы для ввода данных в БД со стороны удаленных пользователей.



## ГЛАВА 11

# Расширение возможностей администрирования сайта «Мир книг» и создание пользовательских форм

В предыдущих главах мы создали сайт «Мир книг», который позволяет пользователям получать из каталога списки книг и авторов. И сейчас каждый посетитель сайта имеет доступ к одним и тем же страницам и типам информации из базы данных.

Однако хотелось бы предоставить пользователю индивидуальные услуги, которые зависят от его предпочтений и предыдущего опыта использования сайта и его настроек. Например, возможность добавлять на страницы сайта свою информацию, что актуально для интернет-магазинов, социальных сетей, блогов и прочих подобных ресурсов.

В этой главе мы расширим возможности нашего сайта «Мир книг», добавив в него счетчик посещений домашней страницы, функционирующий на основе сессий. Этот относительно простой пример покажет, как с помощью сессий реализовать анализ поведения на сайте анонимных пользователей.

Мы также организуем на сайте «Мир книг» поддержку пользовательских аккаунтов и таким образом продемонстрируем аутентификацию, разграничение уровней доступа, сессии и пользовательские формы.

В этой главе будут рассмотрены следующие вопросы:

- сессии в Django;
- создание страниц для авторизации пользователей в Django;
- организация проверки подлинности входа пользователя в систему;
- добавление к сайту страницы для создания заказов на книги;
- формы и управление формами в Django;
- как создать форму для ввода и обновления информации на сайте на основе класса `Form()`;
- как создать форму для ввода и обновления информации на сайте на основе класса `ModelForm()`.

## 11.1. Сессии в Django

Сессии позволяют хранить и получать произвольные данные, запрошенные на основе индивидуального поведения пользователя на сайте. Разберемся, что же такое сессии.

Все взаимодействия между браузерами пользователей и серверами осуществляются при помощи протокола HTTP, который не сохраняет состояние таких взаимодействий (stateless). Это означает, что сообщения между клиентом и сервером являются полностью независимыми друг от друга, т. е. не существует какого-либо представления «последовательности» или поведения системы в зависимости от предыдущих сеансов взаимодействия. Так что если вы хотите создать сайт, который будет отслеживать взаимодействие с клиентом, то вам нужно реализовать это самостоятельно.

Сессии представляют собой механизм, на основе которого Django отслеживает состояние и взаимодействие между сайтом и каким-либо браузером. Сессии позволяют хранить произвольные данные браузера и извлекать их в тот момент, когда между браузером и сайтом устанавливается соединение. Данные доставляются и сохраняются в сессии при помощи соответствующего «ключа».

Django для этого использует файлы cookie. Они содержат специальный идентификатор сессии, который выделяет каждый браузер в соответствующую сессию. Реальные данные сессии по умолчанию хранятся в базе данных сайта. Впрочем, у вас есть возможность настроить Django так, чтобы сохранять данные сессий в других местах (кеше, файлах, «безопасных» cookie). Но все же хранение по умолчанию является хорошей и безопасной возможностью.

Сессии становятся доступны автоматически в тот момент, когда вы создаете «скелет» сайта. Необходимые настройки конфигурации выполняются в разделах `INSTALLED_APPS` и `MIDDLEWARE` файла проекта `WebBooks\WebBooks\settings.py` (рис. 11.1).

Доступ к переменной `session` можно получить в соответствующем представлении через параметр `request` (`HttpRequest` передается как первый аргумент в каждое представление). Переменная сессии обеспечивает связь с определенным пользователем (если быть более точным — с определенным браузером пользователя), который определяется при помощи идентификатора (`id`) сессии, получаемого из файла cookie браузера.

Переменная (или поле) `session` является объектом-словарем, в который можно делать записи неограниченное число раз. С ним вы можете выполнять любые стандартные операции: чтение, очистку всех данных, проверку наличия ключа, циклы по данным и т. п. Далее представлены фрагменты кода, которые показывают, как получать, задавать и удалять некоторые данные при помощи ключа `my_car`, связанного с текущей сессией (браузером):

```
# Получение значения сессии при помощи ключа('my_car').
# Если такого ключа нет, то возникнет ошибка KeyError
my_car = request.session['my_car']

# Получение значения сессии. Если значения не существует,
# то вернется значение по умолчанию ('mini')
my_car = request.session.get('my_car', 'mini')

# Передача значения в сессию
request.session['my_car'] = 'mini'
```

```
# Удаление значения из сессии
del request.session['my_car']
```

Этот API имеет и другие методы, которые большей частью используются для управления файлами cookie, связанными с сессией. Например, существуют методы проверки того, что cookie поддерживаются клиентским браузером, другие методы служат для установки и проверки предельных дат жизни cookie, а также для очистки хранилища от просроченных сессий.

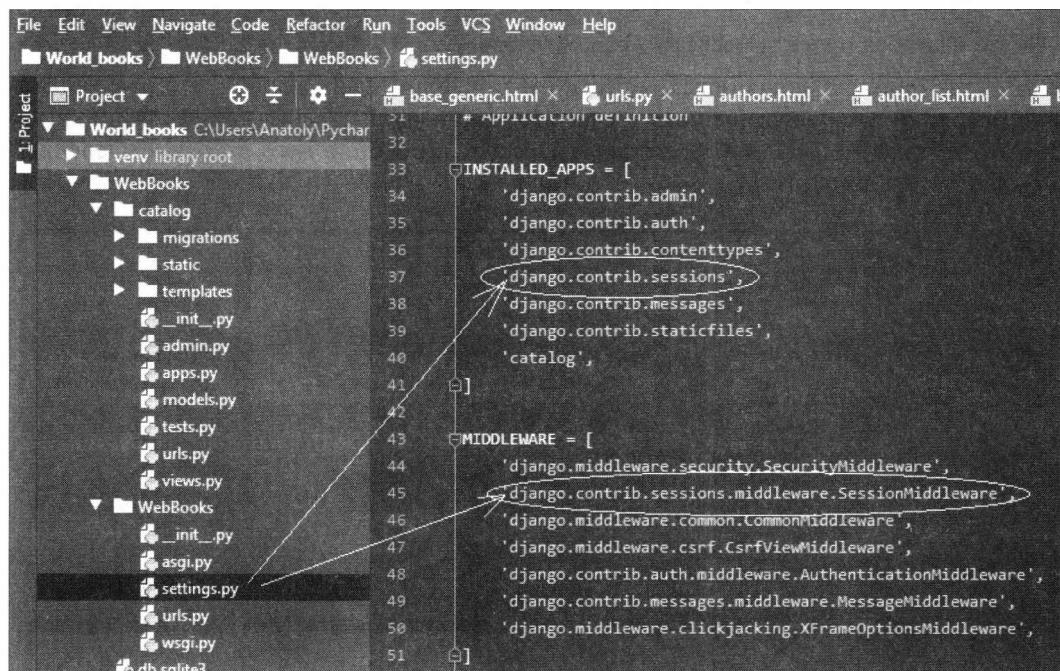


Рис. 11.1. Разделы конфигурирования сессий в файле settings.py

По умолчанию Django сохраняет сессии в базе данных и отправляет соответствующие cookie клиенту только тогда, когда сессия была изменена или удалена. Если вы обновляете какие-либо данные при помощи ключа сессии, как показано в предыдущем фрагменте кода, то вам не нужно беспокоиться о процессе сохранения! Например:

```
# Это присваивание распознается как обновление сессии
# и данные будут сохранены
request.session['my_car'] = 'mini'
```

Если же вы обновляете информацию внутри данных сессии, тогда Django не распознает эти изменения и не сохраняет данные. Например, если вы изменили параметр wheels внутри переменной my\_car, как показано далее, то нужно явно указывать, что сессия была изменена:

```
# Объект сессии модифицируется неявно.
# Изменения НЕ БУДУТ сохранены!
request.session['my_car']['wheels'] = 'alloy'
```

Но можно указать и явное сохранение данных сессии:

```
# Явное указание, что данные изменены.
# Сессия будет сохранена, cookie обновлены (если необходимо)
request.session.modified = True
```

Можно изменить поведение сессий таким образом, чтобы они записывали любое свое изменение в базу данных и отправляли cookie при каждом запросе, путем установки значения

```
SESSION_SAVE_EVERY_REQUEST = True
```

в файле настроек проекта `WebBooks\WebBooks\settings.py`.

В качестве примера использования сессий мы обновим наш сайт так, чтобы сообщать пользователю число совершенных им визитов на главной странице сайта «Мир книг». Для этого откройте файл `WebBooks\catalog\views.py` и добавьте в него строки, выделенные серым фоном в листинге 11.1.

#### Листинг 11.1. Измененный код файла `WebBooks\catalog\views.py`

```
def index(request):
    text_head = 'На нашем сайте вы можете получить книги в электронном виде'
    # Данные о книгах и их количестве
    books = Book.objects.all()
    num_books = Book.objects.all().count()
    # Данные об экземплярах книг в БД
    num_instances = BookInstance.objects.all().count()
    # Доступные книги (статус = 'На складе')
    num_instances_available = BookInstance.objects.filter(
        status_exact=2).count()
    # Данные об авторах книг
    authors = Author.objects
    num_authors = Author.objects.count()
    # Число посещений этого view, подсчитанное в переменной session
    num_visits = request.session.get('num_visits', 0)
    request.session['num_visits'] = num_visits + 1
    # Словарь для передачи данных в шаблон index.html
    context = {'text_head': text_head,
               'books': books, 'num_books': num_books,
               'num_instances': num_instances,
               'num_instances_available': num_instances_available,
               'authors': authors, 'num_authors': num_authors,
               'num_visits': num_visits
    }
    # передача словаря context с данными в шаблон
    return render(request, 'catalog/index.html', context)
```

В первую очередь мы формируем здесь переменную `'num_visits'` из сессии и устанавливаем для нее значение 0 (если оно не было установлено ранее). Затем каждый раз при обращении к задействованному представлению (`view`) мы увеличиваем значение этой переменной на единицу и сохраняем его в сессии (до следующего посещения этой страницы пользователем). В завершение переменная `num_visits` передается в шаблон

главной страницы через переменную context. Для показа значения этой переменной добавьте в шаблон главной страницы сайта несколько строк кода \WebBooks\templates\catalog\index.html из листинга 11.2 (изменения выделены серым фоном).

### Листинг 11.2. Измененный код файла \WebBooks\templates\catalog\index.html

```
<div class="row text-center text-dark lh-2">
    <h3>Книги</h3>
</div>
{%
    if num_books > 0 %
        <div class="row my-2">
            {% for obj in books %}
                <div class="card" style="width: 9rem;">
                    
                    <div class="card-body">
                        <p class="card-text small">
                            {{obj.title}}, Цена:{{obj.price}} руб.
                        </p>
                    </div>
                </div>
            {% endfor %}
        </div>
    {% endif %}
<div class="row text-center text-dark lh-2">
    <h6>Число посещений данной страницы - {{ num_visits }}</h6>
</div>
</div>
{% endblock content %}
```



Рис. 11.2. Счетчик посещения страницы сайта на базе переменной session

Запустим наш сервер и несколько раз выполним вход на главную страницу. Значение числа посещений будет изменяться всякий раз при входе на главную страницу (рис. 11.2).

## 11.2. Аутентификация и авторизация пользователей в Django

В этом разделе мы продемонстрируем возможность входа пользователя на сайт с использованием собственного аккаунта. Кроме того, покажем, как реализовать контроль действий пользователя в зависимости от того, вошел он на сайт с собственным паролем или нет, а также имеет ли он соответствующие права доступа (permissions) к элементам сайта. Для того чтобы показать все это, мы расширим сайт «Мир книг», добавив в него страницы для входа/выхода, а также страницы просмотра/редактирования книг, специфические для пользователя и для персонала.

### 11.2.1. Немного об аутентификации пользователей в Django

Django предоставляет систему аутентификации и авторизации (permission) пользователя, реализованную на основе сессий, с которыми мы познакомились в предыдущем разделе. Система аутентификации и авторизации позволяет проверять учетные данные пользователей и определять, какие действия они могут выполнять. Django имеет встроенные модели для отдельных «Пользователей» и для «Групп» пользователей (более чем один пользователь с одинаковыми правами), а также саму систему прав доступа.

Мы реализуем аутентификацию пользователя путем создания страниц входа/выхода и добавления разграничения доступа к моделям данных, а также продемонстрируем контроль доступа к некоторым страницам. Мы применим авторизацию для показа пользователям и сотрудникам, сопровождающим сайт, списков книг, которые находятся в заказе.

Система аутентификации является очень гибкой и позволяет формировать свои собственные URL-адреса, формы, отображения, а также шаблоны страниц для зарегистрированных пользователей. Но в этом разделе мы воспользуемся встроенными в Django методами показа данных и построения форм аутентификации, а также методами построения страниц входа и выхода. Нам понадобится создать шаблоны соответствующих страниц, но сделать это довольно просто.

Система аутентификации пользователей была подключена к сайту автоматически, когда мы создали его проект (при помощи команды `django-admin startproject`), так что сейчас нам ничего делать не нужно. Таблицы базы данных для пользователей и модели авторизации были созданы, когда мы в первый раз выполнили команду миграции (`python manage.py migrate`). Соответствующие настройки сделаны в параметрах `INSTALLED_APPS` и `MIDDLEWARE` файла проекта `settings.py` (рис. 11.3).

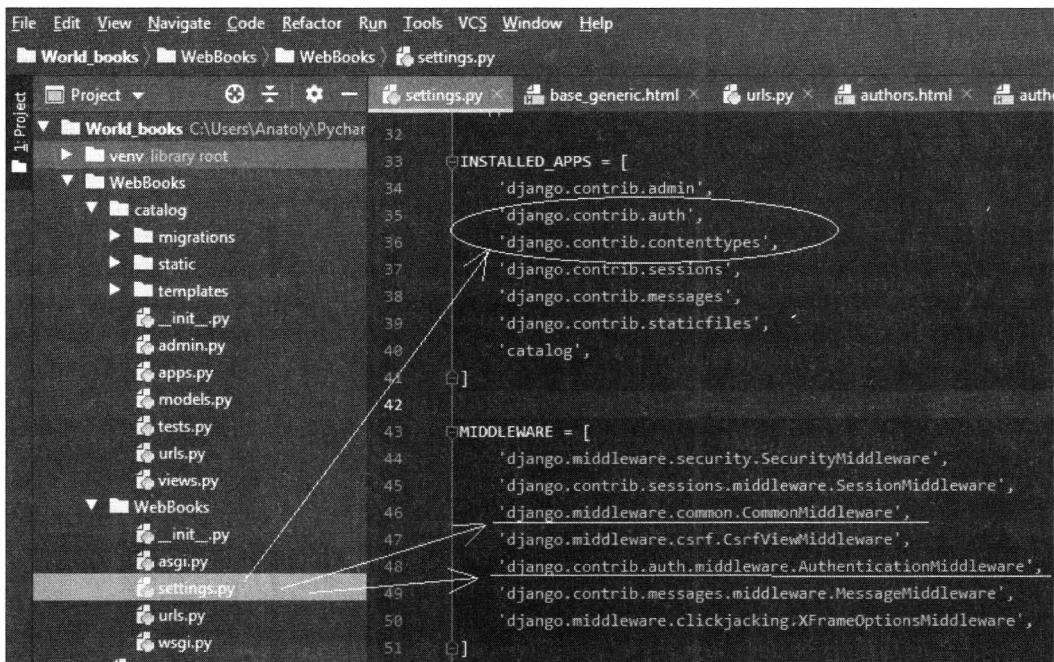


Рис. 11.3. Параметры настройки авторизации пользователей в файле settings.py

## 11.2.2. Создание отдельных пользователей и групп пользователей

Мы уже создали первого пользователя, когда рассматривали административную панель сайта. Это был суперпользователь, созданный при помощи команды `python manage.py createsuperuser`. Наш суперпользователь уже авторизован и имеет все необходимые уровни доступа и к данным, и к функциям. Таким образом, нам необходимо создать тестового пользователя.

Можно создавать пользователей и программным способом. Например, в том случае, если вы разрабатываете интерфейс, который позволяет пользователям создавать их собственные аккаунты (при этом вы не должны предоставлять доступ пользователям к административной панели вашего сайта). Вот образец такого программного кода:

```

from django.contrib.auth.models import User
# Создать пользователя и сохранить его в базе данных
user = User.objects.create_user('myusername',
                                'myemail@crazymail.com',
                                'mypassword')
# Обновить поля и сохранить их снова
user.first_name = 'Михаил'
user.last_name = 'Петров'
user.save()

```

Тем не менее мы создадим группу, а затем пользователя в этой группе с помощью административной панели сайта.

Несмотря на то что у нас пока нет никаких разрешений на добавление к нашему сайту каких-либо пользователей, если мы захотим это сделать в будущем, то будет намного проще добавлять их к уже созданной группе с заданной аутентификацией. И административная панель сайта предоставляет наиболее быстрый способ создания соответствующих групп и аккаунтов.

Итак, запускаем сервер разработки и переходим к административной панели сайта (<http://127.0.0.1:8000/admin/>). Заходим на сайт при помощи параметров аккаунта суперпользователя (имени пользователя и пароля). Самая верхняя страница панели администратора показывает все наши модели. Для того чтобы увидеть записи в разделах **Authentication** и **Authorisation**, вы можете нажать на ссылку **Группы** или **Пользователи** в разделе **ПОЛЬЗОВАТЕЛИ И ГРУППЫ** (рис. 11.4).

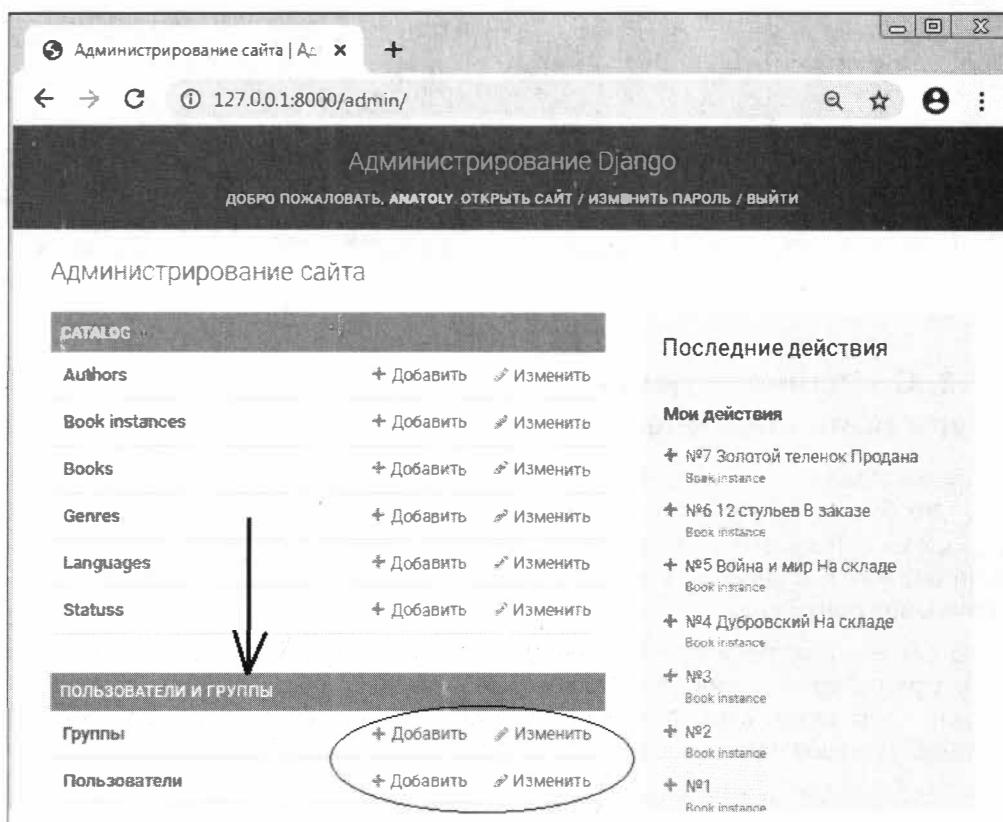


Рис. 11.4. Разделы административной панели сайта для создания групп и пользователей

В первую очередь давайте создадим новую группу — нажмите для этого на ссылку **+Добавить** в строке **Группы**. Введите для нее имя Пользователи сайта (рис. 11.5). Для этой группы не нужны какие-либо разрешения, поэтому просто нажмите кнопку **СОХРАНИТЬ**, и вы перейдете к списку групп.

Теперь давайте создадим пользователя. Для этого возвращаемся на главную страницу административной панели и нажимаем ссылку **+Добавить** в строке **Пользователи**, после чего откроется окно со списком пользователей (рис. 11.6).

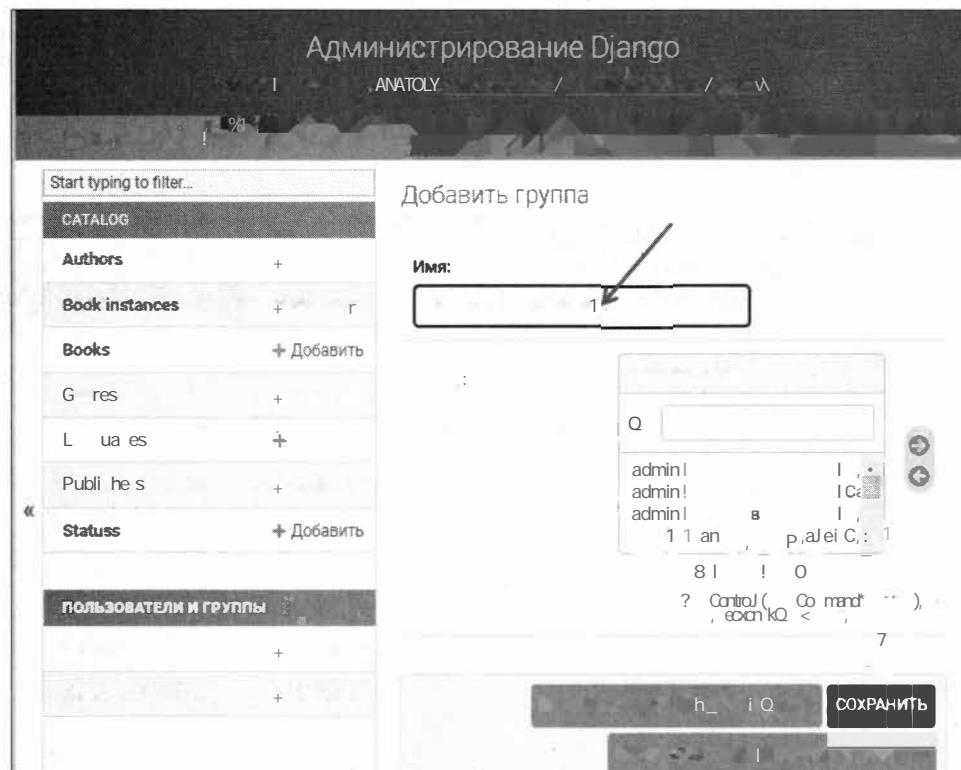


Рис. 11.5. Создание группы пользователей в административной панели сайта

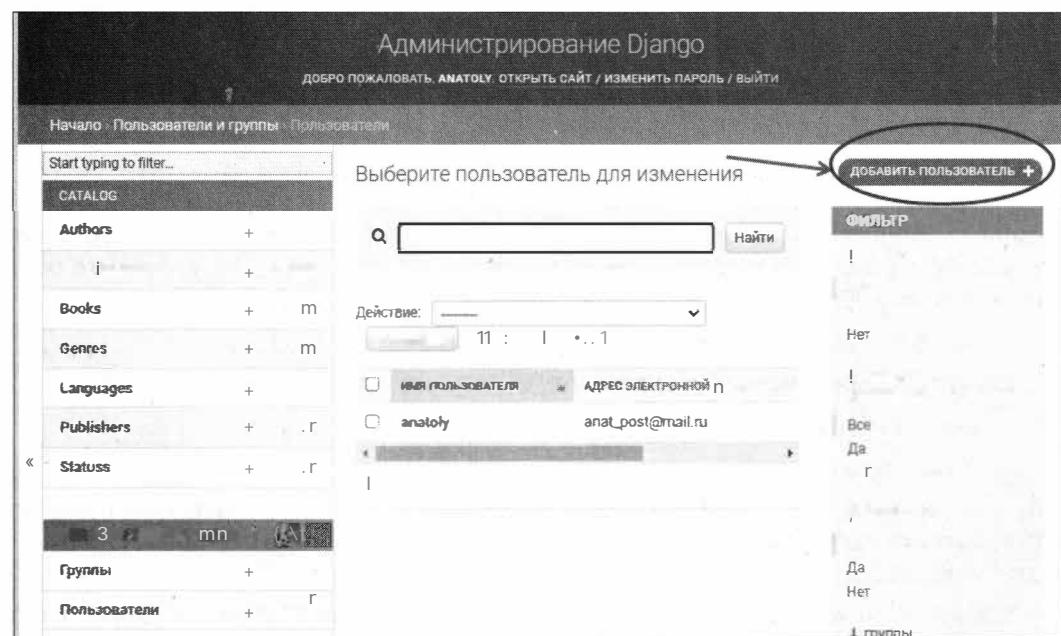


Рис. 11.6. Окно со списком пользователей в административной панели сайта

В этом окне нажмите на кнопку **ДОБАВИТЬ ПОЛЬЗОВАТЕЛЬ +**. Будет выполнен переход к диалоговому окну ввода сведений о новом пользователе (рис. 11.7).

Введите здесь в полях **Имя пользователя**, **Пароль** и **Подтверждение пароля** соответствующие данные для нашего тестового пользователя и нажмите кнопку **СОХРАНИТЬ** для завершения процесса создания пользователя.

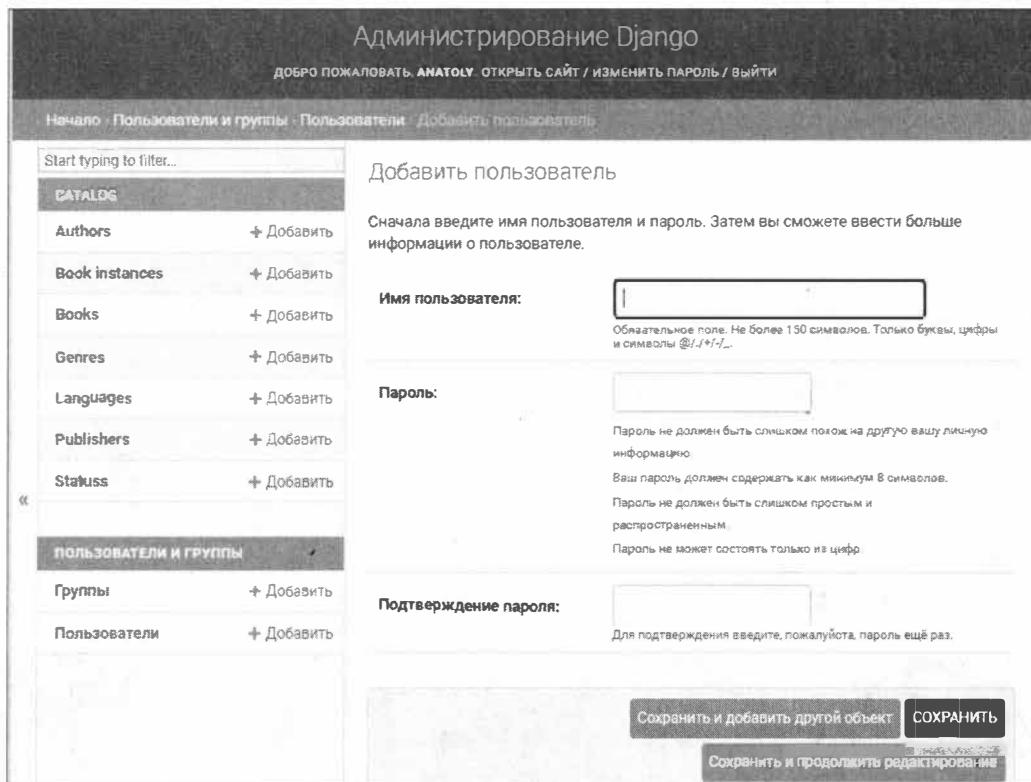


Рис. 11.7. Окно для ввода новых пользователей в административной панели сайта

Административная часть сайта создаст нового пользователя и немедленно перенаправит вас на страницу изменения параметров пользователя (рис. 11.8), где вы сможете ввести для него некоторую персональную информацию:

- изменить имя пользователя (входное имя на сайт);
- добавить имя пользователя (по паспорту);
- добавить фамилию пользователя;
- ввести его электронный адрес.

Как можно видеть, мы задали пользователю имя: **Тестер\_1**. Если прокрутить страницу с параметрами пользователя вниз, то можно увидеть блок, в котором можно задать для него права доступа (рис. 11.9).

В этом блоке можно задать статус пользователя и признак активности (может быть установлен только соответствующий флаг). Кроме того, можно определить группу для пользователя и необходимые параметры доступа.

The screenshot shows the 'User Information' section of the administrative interface. On the left, there's a sidebar with a search bar and links for 'CATALOG' (Authors, Book instances, Books, Genres, Languages, Publishers, Status), 'ПОЛЬЗОВАТЕЛИ И ГРУППЫ' (Группы, Пользователи), and a back arrow. The main area has tabs for 'Изменить пользователя' (Edit user) and 'ИСТОРИЯ' (History). Under 'Изменить пользователя', it shows a user named 'Тестер\_1'. The 'Имя пользователя:' field contains 'Тестер\_1'. A note says: 'Обязательное поле. Не более 150 символов. Только буквы, цифры и символы @/./-/\_.'. The 'Пароль:' field contains a hashed password: 'алгоритм: pbkdf2\_sha256 итерации: 390000 соль: GК3qbU\*\*\*\*\*кшц: 6Vzkg\*\*\*\*\*'. A note below says: 'Пароли хранятся в зашифрованном виде, поэтому нет возможности просмотреть пароль этого пользователя, но вы можете изменить его используя эту форму.' Below this is a 'Персональная информация' tab with fields for 'Имя:' (Name), 'Фамилия:' (Surname), and 'Адрес электронной почты:' (Email address).

Рис. 11.8. Блок с персональными данными пользователя в административной панели сайта

The screenshot shows the 'User Access Rights' section. On the left, there's a sidebar with a search bar and links for 'CATALOG' (Authors, Book instances, Books, Genres, Languages, Publishers, Status), 'ПОЛЬЗОВАТЕЛИ И ГРУППЫ' (Группы, Пользователи), and a back arrow. The main area has tabs for 'Права доступа' (Access rights) and 'История' (History). Under 'Права доступа', there are checkboxes for 'Активный' (Active), 'Статус персонала' (Personnel status), and 'Статус суперпользователя' (Superuser status). There are also sections for 'Группы:' (Groups) and 'Права пользователя:' (User rights). The 'Группы:' section includes a 'Доступные группы' (Available groups) list with a 'Фильтр' (Filter) input, a 'Пользователи сайта' (Website users) list, and buttons for 'Выбрать все' (Select all) and 'Удалить все' (Delete all). The 'Права пользователя:' section includes a 'Доступные права пользователя' (Available user rights) list with a 'Фильтр' (Filter) input, a list of rights like 'admin | запись в журнале | Статистика', and buttons for 'Выбрать все' (Select all) and 'Удалить все' (Delete all). A note at the bottom says: 'Индивидуальные права данного пользователя. Удерживайте "Control" (или "Command" на Mac), чтобы выбрать несколько значений.'

Рис. 11.9. Блок задания прав доступа пользователей в административной панели сайта

В самой нижней части страницы изменения параметров пользователя имеется блок **Важные даты**, в котором можно внести даты, относящиеся к этому пользователю (дату подключения к сайту и дату последнего входа), а также расположена кнопочная панель для сохранения введенных изменений и перехода на другие страницы (рис. 11.10).

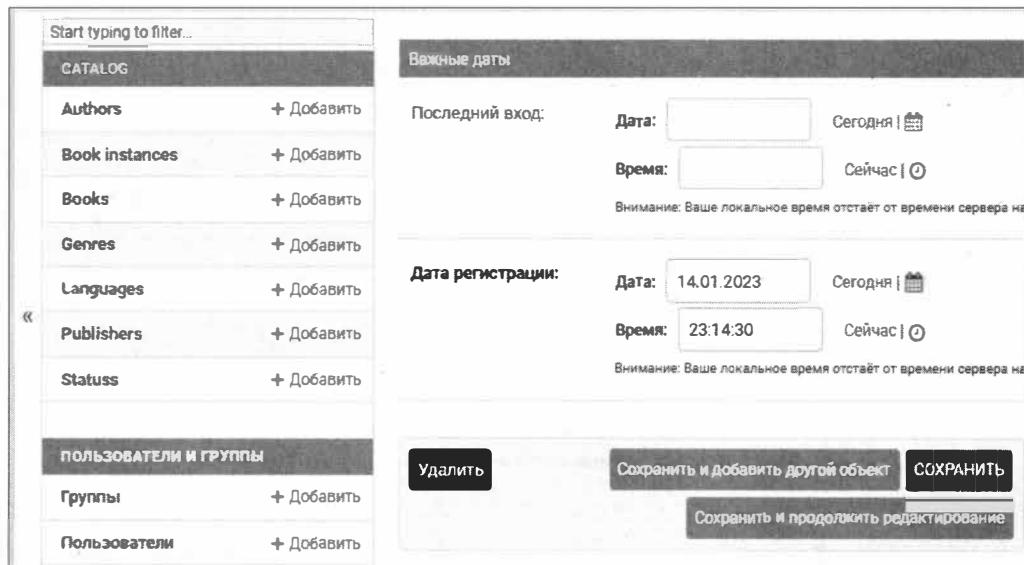


Рис. 11.10. Блок Важные даты и кнопочная панель в административной панели сайта

Поскольку это будет обычный внешний пользователь, не сотрудник компании, то мы здесь не будем ничего менять, а просто нажмем кнопку **СОХРАНИТЬ**. После этого в системе появится новый пользователь с именем **Тестер\_1** (рис. 11.11).

The screenshot shows the Django admin interface for managing users. The top bar says 'Администрирование Django' and 'ДОБРО ПОЖАЛОВАТЬ, ANATOLY'. The sidebar shows 'CATALOG' and 'ПОЛЬЗОВАТЕЛИ И ГРУППЫ' sections. The main area shows a message: 'Пользователь "Тестер\_1" был успешно изменен.' (User "Tester\_1" was successfully changed.) Below this is a search bar with 'Найти' (Find) button and a table with columns: 'Действие:' (Action), 'Имя пользователя' (User name), 'Адрес электронной почты' (Email address), 'Имя' (First name), 'Фамилия' (Last name), and 'Статус персонала' (Personnel status). Two users are listed: 'anatoly' (email: anat\_post@mail.ru, first name: Anatoly, last name: Postnikov, personnel status: Tester) and 'Тестер\_1' (email: tmyrtp@mail.ru, first name: Тестер, last name: Тестер, personnel status: Tester). A callout bubble points to the 'Тестер\_1' row with the text 'Новый пользователь' (New user). The bottom of the table shows '2 пользователя' (2 users).

Рис. 11.11. Новый пользователь в окне административной панели Django

Вот и все! Теперь у нас есть учетная запись «обычного пользователя» сайта, которую можно использовать для тестирования работы сайта (после того, как мы добавим страницы входа пользователей в систему).

### 11.2.3. Создание страницы регистрации пользователя при входе на сайт

Django предоставляет почти все, что нужно для создания страниц аутентификации входа-выхода из системы и управления паролями. Этот набор включает в себя URL-адреса, представления (views) и формы, но не содержит шаблоны (придется создать свой собственный шаблон самостоятельно).

#### ПРИМЕЧАНИЕ

Мы можем поместить страницы аутентификации, включая URL-адреса и шаблоны, в папку catalog нашего единственного приложения. Однако при наличии нескольких приложений лучше отделить вход в систему и обеспечить доступность страницы авторизации пользователей для всего сайта. Именно так мы и поступим.

Начнем с создания ссылок на URL-адреса. Добавьте в нижнюю часть файла проекта WebBooks\WebBooks\urls.py код листинга 11.3 (изменения выделены серым фоном).

#### Листинг 11.3. Измененный код файла WebBooks\WebBooks\urls.py

```
from django.contrib import admin
from django.urls import include, path
# добавлено для работы с медиафайлами
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('', include('catalog.urls')),
    path('admin/', admin.site.urls),
]
# добавлено для работы с медиафайлами локально
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
                         document_root=settings.MEDIA_ROOT)
# добавлено для регистрации входа пользователей
urlpatterns += [
    path('accounts/', include('django.contrib.auth.urls')),
]
```

Следующий шаг — создайте каталог (папку) registration в папке templates. Затем в эту папку добавьте шаблон — файл login.html (рис. 11.12).

Затем в файле login.html напишите код листинга 11.4.

#### Листинг 11.4. Измененный код файла WebBooks\templates\registration\login.html

```
{% extends "catalog/base.html" %}
{% block content %}
```

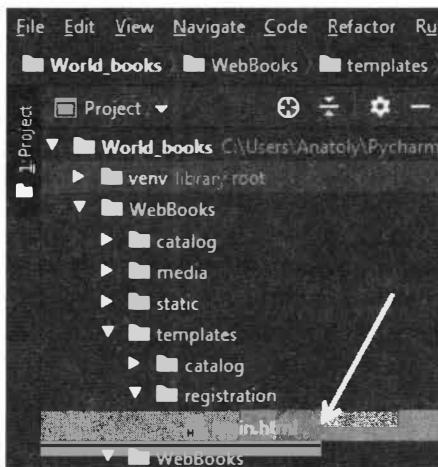


Рис. 11.12. Создание каталога registration в папке templates и файла login.html

```
% if form.errors %}
<p>Ваше имя пользователя и пароль не совпадают.  
Пожалуйста, попробуйте еще раз.</p>
{% endif %}
{% if next %}
  {% if user.is_authenticated %}
    <p>Вы не имеете доступа к этой странице.  
Войдите в систему с другими параметрами.</p>
  {% else %}
    <p>Войдите в систему, чтобы увидеть эту страницу.</p>
  {% endif %}
{% endif %}
<form method="post" action="{% url 'login' %}">
  {% csrf_token %}
  <div class="row my-2 text-center">
    <div class="col-4">
      <table class="table table-bordered my-2 text-start">
        <tr>
          <td>{{ form.username.label_tag }}</td>
          <td>{{ form.username }}</td>
        </tr>
        <tr>
          <td>{{ form.password.label_tag }}</td>
          <td>{{ form.password }}</td>
        </tr>
      </table>
    </div>
    <div class="col-8"></div>
  </div>
  <div class="row my-2 text-start">
    <div class="col-4">
      <input type="submit" value="Вход" />
      <input type="hidden" name="next" value="{{ next }}" />
    </div>
  </div>
</form>
```

```

{# Предполагается, что вы настроили #}
{# представление password_reset в своем URLconf#}
<p><a href="{% url 'password_reset' %}"> Утерян пароль?</a></p>
</div>
</div>
<div class="col-8"></div>
</form>
{% endblock %}

```

Этот шаблон похож на шаблоны, которые мы создавали раньше. Он расширяет наш базовый шаблон и переопределяет блок контента. Остальная часть кода — это довольно-таки стандартная процедура обработки формы, о которой мы поговорим в следующем разделе. Все, что вам нужно сделать, — это показать форму, в которую можно ввести имя пользователя и пароль, а если будут введены недопустимые значения, предложить (когда страница обновится) ввести правильные.

Если теперь перейти по адресу <http://127.0.0.1:8000/accounts/login/>, то вы должны увидеть страницу входа с полями идентификации пользователя (рис. 11.13).

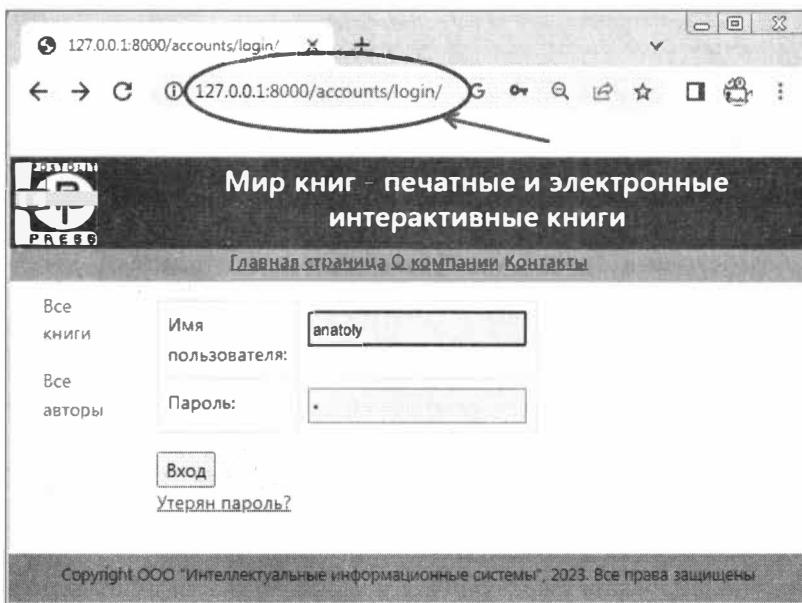


Рис. 11.13. Страница входа с идентификацией пользователя

Введите здесь входные данные пользователя, и вы будете перенаправлены на другую страницу сайта — по умолчанию это страница профиля пользователя по адресу <http://127.0.0.1:8000/accounts/profile/>. Но поскольку страницу с профилем мы еще не сделали, то вам будет выдано сообщение об ошибке с информацией, что страница `profile` отсутствует (рис. 11.14).

Ликвидируем этот недочет. Откройте настройки проекта (файл `settings.py`) и добавьте в конец файла код листинга 11.5.

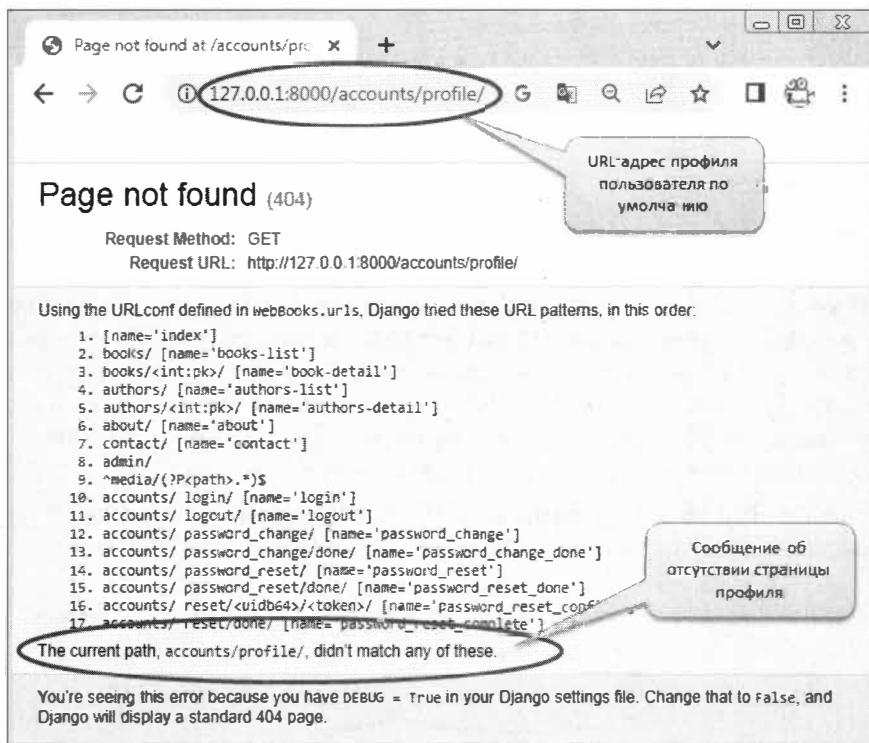


Рис. 11.14. Сообщение об отсутствии страницы profile

#### Листинг 11.5. Измененный код файла WebBooks\WebBooks\settings.py

```
# Переадресация на главную страницу сайта после входа в систему
LOGIN_REDIRECT_URL = '/'
```

Теперь, если повторить вход в систему, то ошибка больше выдаваться не будет и пользователь будет успешно перенаправлен на домашнюю страницу сайта.

Если же вы перейдете по URL-адресу выхода <http://127.0.0.1:8000/accounts/logout/>, то увидите, что пользователь выведен из системы, но это (по умолчанию) будет показано в окне панели администратора сайта (рис. 11.15).

Но это не то, что нам нужно, такое окно должны получать только пользователи, у которых есть статус `is_staff`, т. е. персонал, обслуживающий сайт. Чтобы этого не происходило, создадим шаблон страницы выхода из системы (файл `templates\registration\logged_out.html`) и внесем в этот файл код листинга 11.6.

#### Листинг 11.6. Измененный код файла WebBooks\templates\registration\logged\_out.html

```
{% extends "catalog/base.html" %}
{% block content %}
<p>Выход из системы!</p>
<a href="{% url 'login'%}"> Нажмите на эту ссылку для входа в систему</a>
{% endblock %}
```

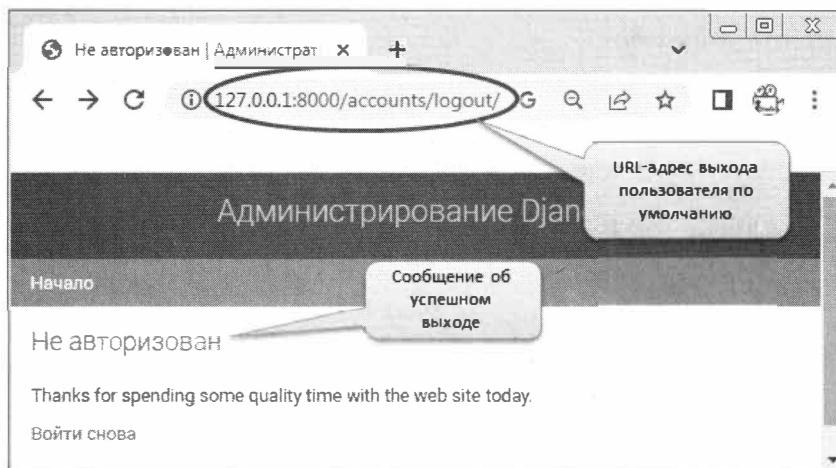


Рис. 11.15. Окно с сообщением об успешном выходе из системы панели администратора сайта

Этот шаблон очень прост. Он выводит сообщение о том, что вы вышли из системы, и предоставляет ссылку, которую вы можете нажать, чтобы вернуться на страницу входа в систему.

Попробуем снова перейти на страницу выхода из системы. Если вы опять видите страницу выхода из системы панели администратора сайта (см. рис. 11.15) вместо своей собственной страницы выхода, то пугаться не стоит. Попробуйте изменить настройку `INSTALLED_APPS` вашего проекта в файле `settings.py` и убедитесь, что строка `'django.contrib.admin'` стоит после строки с вашим приложением `'catalog'` (рис. 11.16).

```

settings.py
INSTALLED_APPS = [
    'catalog',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]

```

The screenshot shows a code editor with three tabs: `settings.py`, `logged_out.html`, and `login.htm`. The `settings.py` tab is active, displaying Python code. An arrow points from the text "Правильный порядок описания приложений в настройках" to the line where `'catalog'` is listed under `INSTALLED_APPS`.

Рис. 11.16. Правильный порядок описания приложений в настройках `INSTALLED_APPS`

Если строка `'django.contrib.admin'` стоит выше, чем строка `'catalog'`, то поменяйте их местами. Дело в том, что оба шаблона расположены по одному и тому же относительному пути, и загрузчик шаблонов Django воспользуется первым найденным.

Если все было сделано правильно, то при запросе по адресу `http://127.0.0.1:8000/accounts/logout/` будет загружена страница на основе вашего шаблона `logged_out.html` (рис. 11.17).

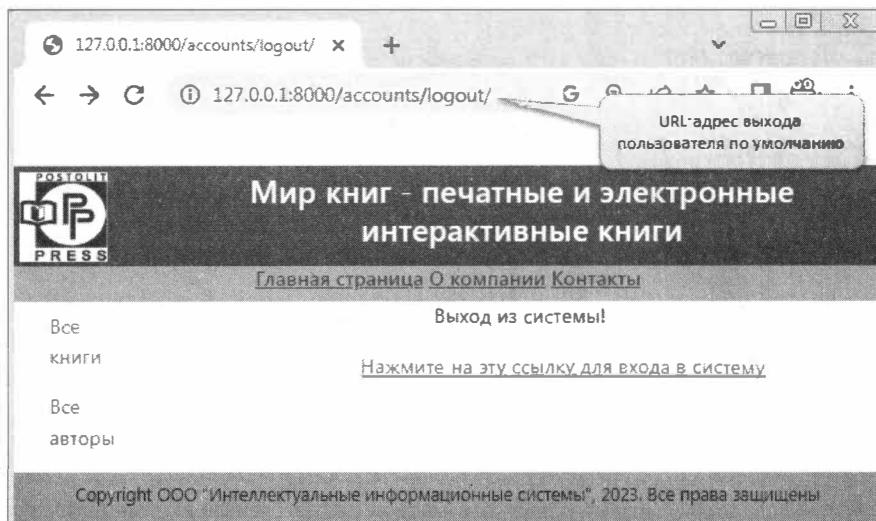


Рис. 11.17. Страница успешного выхода из системы на основе шаблона `logged_out.html`

## 11.2.4. Создание страницы для сброса пароля пользователя

Довольно часто пользователи забывают свой пароль входа на тот или иной сайт. Для таких забывчивых пользователей необходимо предусмотреть возможность восстановления доступа к своему аккаунту путем смены пароля. Система сброса пароля по умолчанию использует электронную почту, чтобы отправить пользователю ссылку на сброс пароля. Для этого на сайте необходимо создать форму, чтобы получить адрес электронной почты пользователя, отправить ему электронное письмо, разрешить ввести новый пароль и сделать отметку о завершении процесса.

Для этих целей создадим в каталоге `\templates\registration\` новую форму (шаблон) `password_reset_form.html` (рис. 11.18).

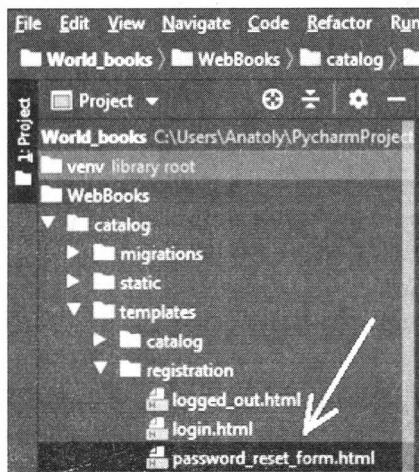


Рис. 11.18. Создание файла шаблона (формы) для сброса пароля пользователя

В созданный файл внесем код листинга 11.7.

#### Листинг 11.7. Измененный код файла WebBooks\templates\registration\password\_reset\_form.html

```
{% extends "catalog/base.html" %}  
{% block content %}  
<form action="" method="post">{% csrf_token %}  
{% if form.email.errors %} {{ form.email.errors }} {% endif %}  
    <div class="row my-2 text-center">  
        <div class="col-6">  
            <table class="table table-bordered my-2 text-start">  
                <td>  
                    <label>Ваш e-mail</label><br>  
                </td>  
                <td>  
                    <p>{{ form.email }}</p>  
                </td>  
            </table>  
        </div>  
        <div class="col-6"></div>  
    </div>  
    <div class="row my-2 text-start">  
        <div class="col-6">  
            <input type="submit" class="btn btn-primary" value="Сброс пароля" />  
        </div>  
        <div class="col-6"></div>  
    </div>  
</form>  
{% endblock %}
```

Затем создадим в каталоге \templates\registration\ форму password\_reset\_done.html (рис. 11.19), которая будет показана пользователю после того, как он введет адрес электронной почты, и внесем в нее код листинга 11.8.

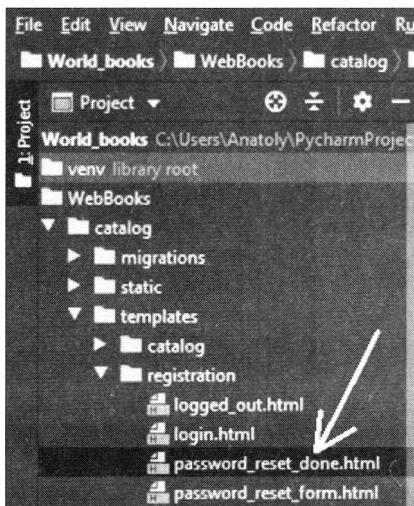


Рис. 11.19. Создание файла шаблона (формы) для смены пароля пользователя

### **Листинг 11.8. Измененный код файла WebBooks\templates\registration\password\_reset\_done.html**

```
{% extends "catalog/base.html" %}  
{% block content %}  
<p>Мы отправили вам по электронной почте инструкции  
по установке пароля. Если они не получены в течение  
нескольких минут, то проверьте папку со спамом.</p>  
{% endblock %}
```

Следующий шаблон представляет собой текст письма, которое мы отправим пользователю по электронной почте, — в нем содержится ссылка на сброс пароля. Для него в том же каталоге `\templates\registration\` создадим файл `password_reset_email.html` и внесем в него код листинга 11.9.

**Листинг 11.9. Измененный код файла WebBooks\templates\registration\password\_reset\_email.html**

Для смены пароля пользователя с электронной почтой {{ email }}  
перейдите по ссылке ниже:  
{{ protocol}}://{{ domain }}% url 'password\_reset\_confirm'  
uidb64=uid token=token %}

И это еще не все — теперь необходимо дать пользователю возможность ввести новый пароль. Нажав на ссылку в полученном электронном письме, пользователь должен попасть на страницу сайта, где сможет сменить пароль. Чтобы предоставить ему такую возможность, в том же каталоге `\templates\registration\` создадим шаблон `password_reset_confirm.html` и внесем в него код листинга 11.10.

**Листинг 11.10. Измененный код файла**  
**WebBooks\templates\registration\password\_reset\_confirm.html**

```
<{% extends "catalog/base.html" %}>
{% block content %}
    {% if validlink %}
        <p>Пожалуйста, введите (и подтвердите) свой новый пароль.</p>
        <form action="" method="post">
            {% csrf_token %}
            <table>
                <tr>
                    <td>{{ form.new_password1.errors }}<br>
                        <label for="id_new_password1">
                            Новый пароль:<br>
                        </label></td>
                    <td>{{ form.new_password1 }}</td>
                </tr>
                <tr>
                    <td>{{ form.new_password2.errors }}<br>
                        <label for="id_new_password2"> Подтвердите пароль:</label></td>
                    <td>{{ form.new_password2 }}</td>
                </tr>
        </form>
    {% endif %}
</div>
```

```

<tr>
    <td></td>
    <td><input type="submit" value=" Сменить пароль " /></td>
</tr>
</table>
</form>
{%
else %
    <h1> Ошибка при смене пароля!!! </h1>
    <p> Ссылка на сброс пароля была недействительной, возможно, потому, что
        она уже использовалась. Пожалуйста, запросите новый сброс пароля.</p>
{% endif %}
{% endblock %}

```

Ну и, наконец, создадим в том же каталоге `\templates\registration\` последний шаблон `password_reset_complete.html`, с помощью которого уведомим пользователя об успешном завершении смены пароля, и внесем в него код листинга 11.11.

#### **Листинг 11.11. Измененный код файла WebBooks\templates\registration\password\_reset\_complete.html**

```

{% extends "catalog/base.html" %}
{% block content %}
<h1> Пароль был успешно изменен!</h1>
<p><a href="{% url 'login' %}">Повторить вход в систему?</a></p>
{% endblock %}

```

Теперь вы сможете проверить функцию смены пароля по ссылке на странице входа в систему.

#### **ПРИМЕЧАНИЕ**

Имейте в виду, что Django отправляет электронные письма для смены пароля только на те адреса пользователей, которые хранятся в базе данных в его аккаунте!

Следует также отметить, что система смены пароля требует, чтобы ваш сайт поддерживал работу с электронной почтой, эти настройки будут сделаны немного позже. Так что эта часть сайта пока еще не будет работать полноценно, однако разработчик имеет возможность протестировать работу всех созданных шаблонов страниц. Чтобы разрешить их тестирование, поместите в конец файла `settings.py` строку из листинга 11.12.

#### **Листинг 11.12. Измененный код файла WebBooks\WebBooks\settings.py**

```

# настройки отправки e-mail
''' это пробная отправка на консоль '''
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'

```

Это обеспечит имитацию отправки электронных писем по e-mail. В процессе разработки сайта данный прием обеспечит отправку электронного письма на консоль. Такое письмо можно будет открыть и посмотреть в терминале PyCharm. Как отправить письмо на реальный электронный адрес, будет показано в следующем разделе.

Проверим работу созданных нами шаблонов и форм. Для этого запустим наш сайт и войдем на страницу входа пользователя <http://127.0.0.1:8000/accounts/login/> (рис. 11.20).

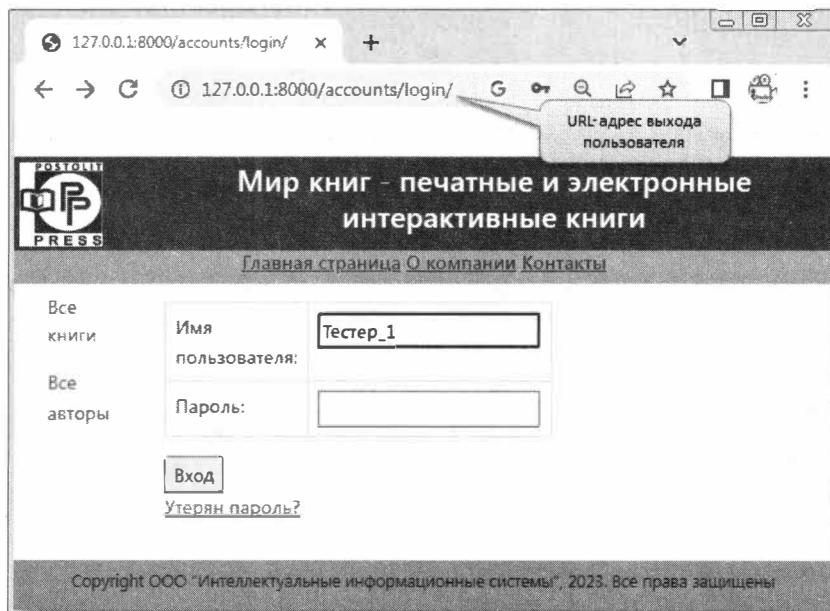


Рис. 11.20. Страница входа в аккаунт пользователя

На этой странице щелкнем левой кнопкой мыши на ссылке **Утерян пароль?**. Вам будет выдана страница с предложением ввести свой электронный адрес для смены пароля (рис. 11.21).

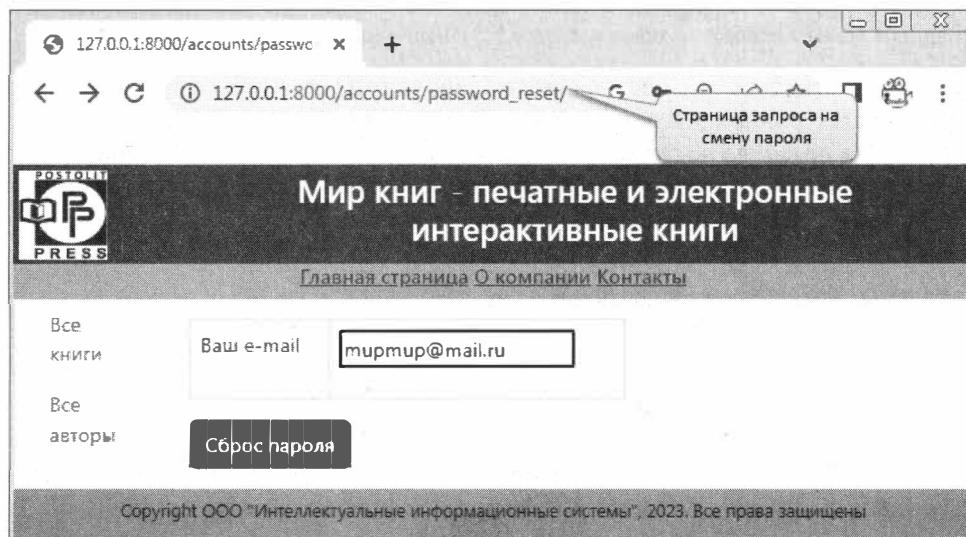


Рис. 11.21. Страница ввода электронного адреса пользователя для смены пароля

Введем электронный адрес пользователя и нажмем на кнопку **Сброс пароля** — появится страница с сообщением, что пользователю на его электронный адрес отправлено письмо с инструкциями по смене пароля (рис. 11.22).

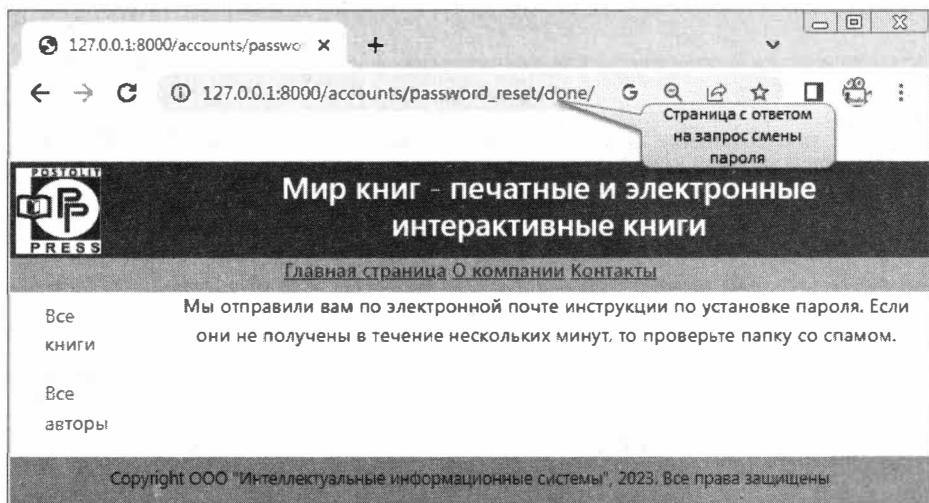


Рис. 11.22. Страница, сообщающая пользователю об отправке ему инструкции по смене пароля

Поскольку наш сайт еще не настроен на отправку электронной почты с внешнего сервера на реальный адрес, то в процессе тестирования работы приложения электронное письмо со ссылкой на страницу для смены пароля будет показано в окне терминала PyCharm (рис. 11.23).

The terminal window shows the following email message:

```

Terminal: Local + [19/Jan/2023 15:28:49] "GET /accounts/password_reset/ HTTP/1.1" 200 3347
Content-Type: text/plain; charset="utf-8"
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Subject: =?utf-8?b?0KHQsdGA0L7RgSDQv9Cw0YDQvtC70Y8g0L3QsCAxMjcuMC4wLjE60DAw?
From: webmaster@localhost
To: mirmir@mail.ru
Date: Thu, 19 Jan 2023 15:29:03 -0000
Message-ID: <167414214315.6692.17479110806774752386@Anatoly-PK>

Для смены пароля пользователя с электронной почтой mirmir@mail.ru перейдите по ссылке ниже:
http://127.0.0.1:8000/accounts/reset/Mg/bia5of-239e839e6acf5d9c30ec0e972d2cf82f/

```

A callout bubble points to the URL in the message body, labeled "Ссылка на страницу для смены пароля".

Рис. 11.23. Инструкция для смены пароля, отправленная пользователю по электронной почте

Щелкните левой кнопкой мыши на ссылке, представленной в полученной пользователем инструкции в окне терминала PyCharm, — вы вернетесь на наш сайт, где будет загружена страница для ввода нового пароля (рис. 11.24).

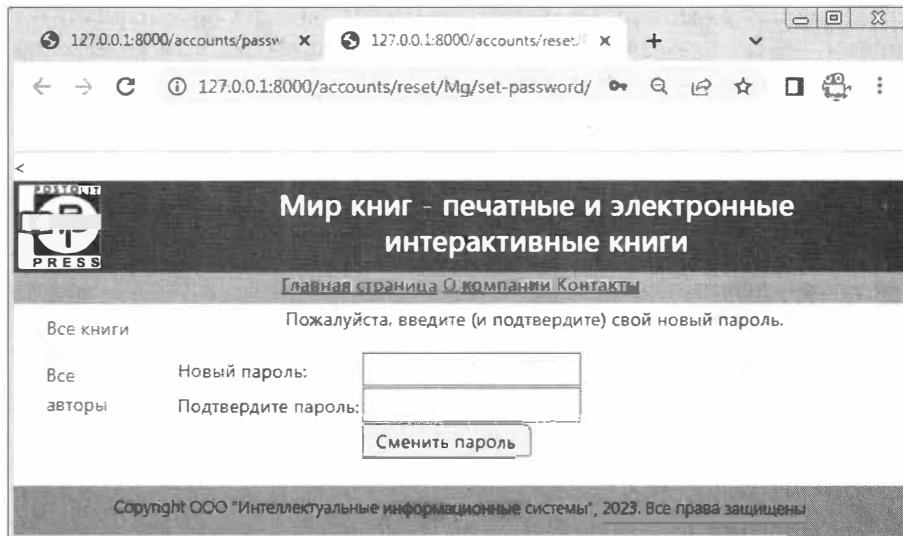


Рис. 11.24. Страница для смены пароля пользователя

На этой странице введем новый пароль, его подтверждение и нажмем на кнопку **Сменить пароль**. Если все было сделано правильно, то откроется окно с сообщением об успешной смене пароля (рис. 11.25). Нажмите в этом окне на ссылку **Повторить вход в систему?**, и вы вернетесь на сайт «Мир книг».

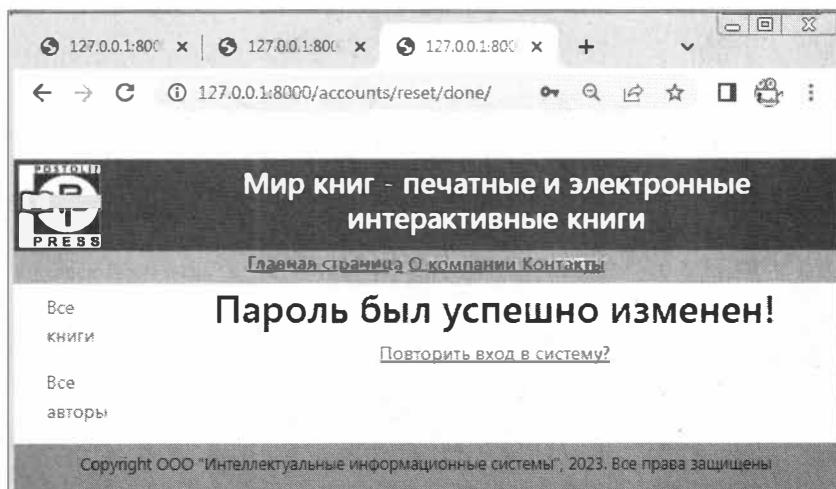


Рис. 11.25. Страница с сообщением об успешной смене пароля

В базе данных пароли пользователей хранятся в зашифрованном виде, и даже главный администратор сайта не может их знать. В этом можно убедиться, открыв с помощью менеджера SQLite таблицу БД с аккаунтами пользователей (рис. 11.26).

При нажатии на ссылку **Повторить вход в систему** пользователь снова будет перенаправлен на страницу входа (рис. 11.27).

id	password	last_login	is_superuser	username
1	pbkdf2_sha256\$390000\$c1yKqMaFy1s1mG7aTAP...G\$sr5Y...	2023-01-15 13:17:57.371616	1	анатолий
2	pbkdf2_sha256\$390000\$6oCm8nehJwomicaVvmANW9i\$...+h8eViuCvmEPdhHjrYlsRZNesIDb8pSWPjZIXefwAwI=	2023-01-15 16:03:02.077170	0	Тестер_1

Рис. 11.26. Зашифрованные пароли в базе данных сайта

Мир книг - печатные и электронные  
интерактивные книги

Главная страница О компании Контакты

Все книги Имя пользователя: Тестер\_1

Все авторы Пароль: .....  
  
Вход Утерян пароль?

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены

Рис. 11.27. Страница входа в систему после смены пароля

Мир книг - печатные и электронные  
интерактивные книги

Главная страница О компании Контакты

На нашем сайте вы можете получить книги в  
электронном виде

Все книги Добро пожаловать на сайт Мир книг. Это очень  
простой веб-сайт, написанный на Django и Bootstrap.  
Разработан на Python в качестве учебного примера с  
использованием PyCharm.

Все авторы Сведения из базы данных

Количество книг - 5      На складе в наличии - 9      Количество авторов - 5

Вход Книги

Книги

Рис. 11.28. Главная страница сайта после входа на нее зарегистрированного пользователя

Если теперь на данной странице набрать новый пароль и нажать на кнопку **Вход**, то будет выведена главная страница сайта (рис. 11.28).

Как видно из данного рисунка, на странице нет никаких признаков того, что на ней находится зарегистрированный пользователь. Кроме того, на ней нет ссылки на страницу с идентификацией пользователя и входа в систему. Доработаем наш сайт: создадим на главной странице сайта ссылки **Вход**, **Выход** и добавим вывод сведений о пользователе, который вошел на сайт после успешной проверки параметров входа.

## 11.3. Настройка почты для отправки сообщения о смене пароля на реальный электронный адрес

Для того чтобы отладить работу модулей идентификации пользователей, мы сымитировали отправку почтового сообщения о возможности смены пароля. Настроим теперь наш проект на отправку электронных писем на реальные электронные адреса. Отправлять сообщения будем через почтовую службу *gmail*. Для этого необходимо создать электронный адрес для почтового клиента и внести некоторые изменения в настройки аккаунта.

Итак, открываем почтовую службу *gmail* и создаем пользователя с условным электронным адресом *my\_address@gmail.com*. Это будет адрес отправителя электронных писем. Теперь входим в настройки аккаунта этого почтового клиента. Здесь в панели **Безопасность** нас интересуют две вкладки: **Двухэтапная идентификация** и **Пароли приложений** (рис. 11.29).

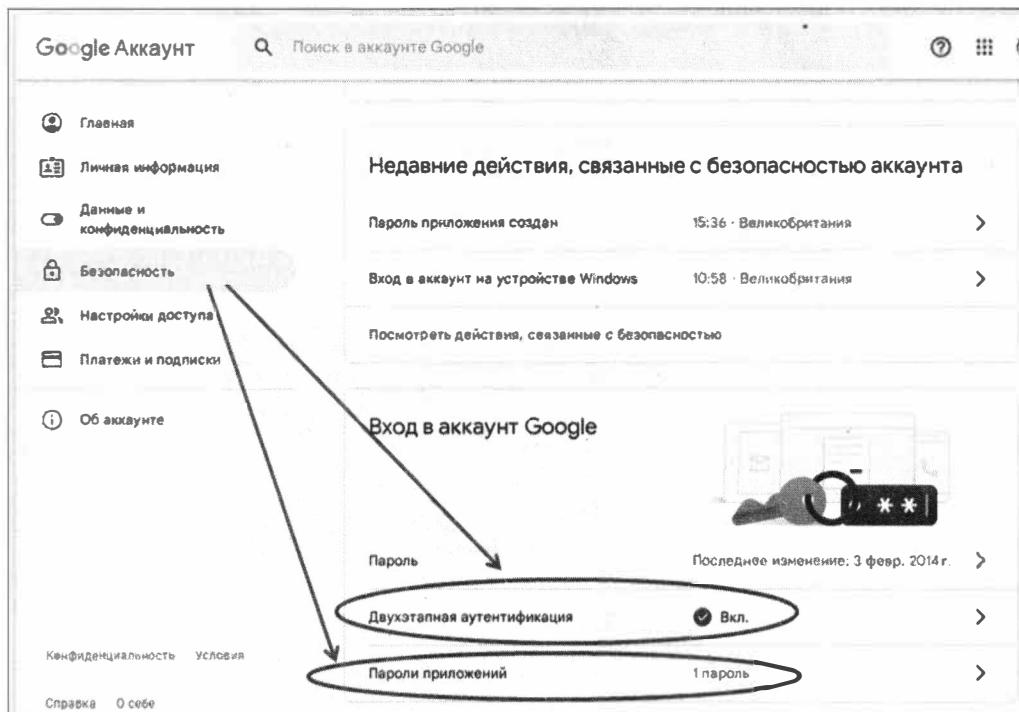


Рис. 11.29. Изменение настроек аккаунта почтового клиента

Сначала активируем свойство двухэтапной аутентификации. Затем открываем вкладку **Пароли приложений** (рис. 11.30).

На вкладке **Пароли приложений** нужно выполнить два действия: выбрать приложение и выбрать устройство.

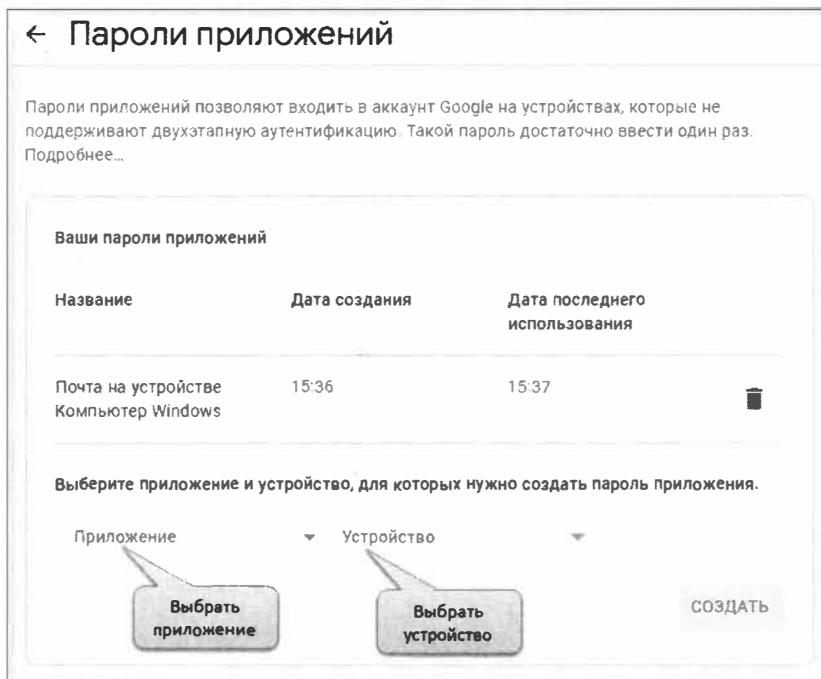


Рис. 11.30. Вкладка Пароли приложений

Сначала открываем список приложений и из этого списка выбираем приложение **Почта** (рис. 11.31).

Затем открываем список устройств и из этого списка выбираем устройство **Компьютер Windows** (рис. 11.32).

Для того чтобы сгенерировать пароль для приложений, нужно нажать на кнопку **Создать** (рис. 11.33).

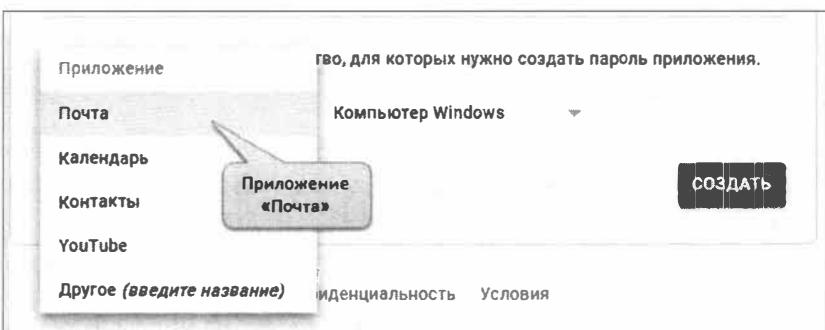


Рис. 11.31. Выбор приложения Почта



Рис. 11.32. Выбор устройства Компьютер Windows

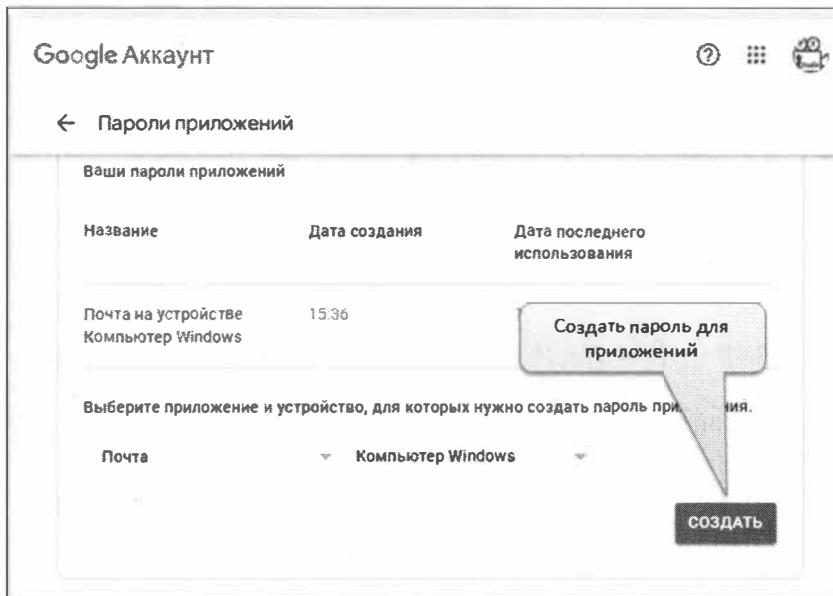


Рис. 11.33. Кнопка для создания пароля для приложений

После этого будет сгенерирован уникальный пароль для приложения (рис. 11.34).

Этот пароль нужно запомнить (а лучше записать), он будет использован вашим сайтом для отправки почтовых сообщений.

Мы сделали самое сложное — выполнили настройку почтового клиента, который будет обслуживать почтовые отправления с нашего сайта. Теперь нужно внести минимальные изменения в настройки нашего приложения. Открываем модуль `settings.py` и вносим в него корректизы, приведенные в листинге 11.13.

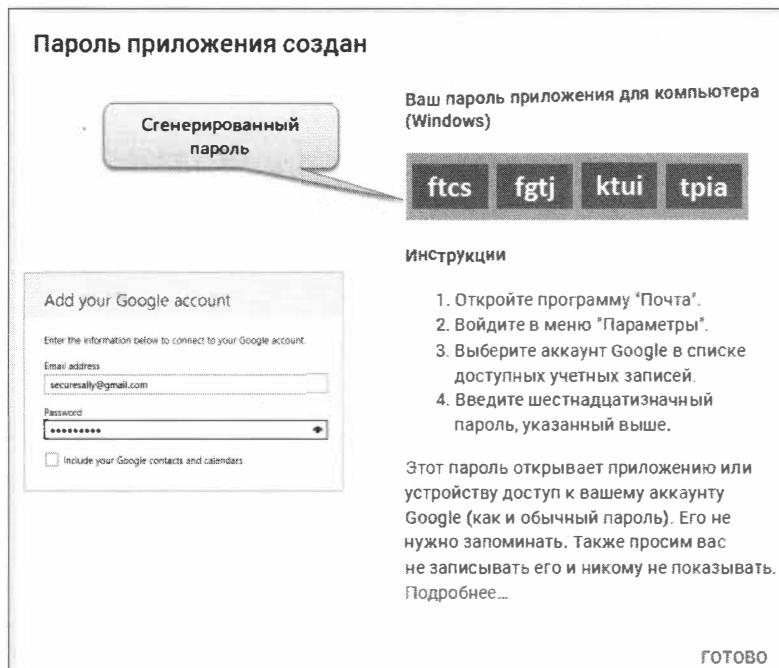


Рис. 11.34. Уникальный пароль для приложений

**Листинг 11.13. Измененный код файла WebBooks\WebBooks\settings.py**

```
# настройки отправки e-mail
''' это пробная отправка на консоль '''
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
''' это реальная отправка через smtp.gmail '''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
# EMAIL_USE_TLS = True
EMAIL_USE_SSL = True # для отправки с gmail
EMAIL_PORT = 465
EMAIL_HOST_USER = "my_address@gmail.com" # от кого
EMAIL_HOST_PASSWORD = "ftcsfgtjktuitpia" # пароль почты отправителя
```

Здесь сначала нужно закомментировать (или удалить) строку, которая инициировала отправку почтовых сообщений в консоль (в окно терминала PyCharm). После этого добавить строки с настройками отправки почтовых сообщений на реальные адреса. Обратите внимание, что почтовая служба gmail использует SSL-протокол безопасности соединений.

После этих изменений вернемся на наш сайт на страницу входа, запросим смену пароля и нажмем на кнопку **Сброс пароля** (рис. 11.35).

Если все действия по настройке были выполнены правильно, то сообщение уйдет на почтовый адрес, указанный в поле **Ваш e-mail**. Открыв полученное письмо, мы получим следующее сообщение (рис. 11.36).

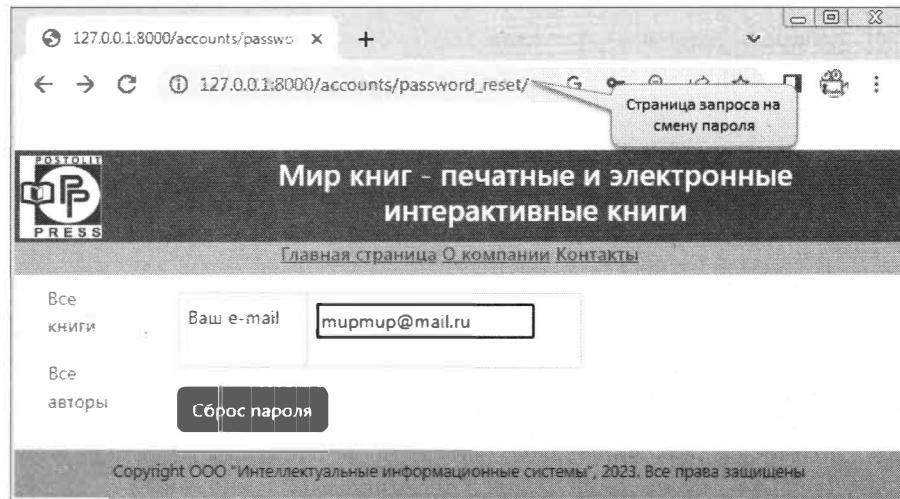


Рис. 11.35. Кнопка активации функции сброса пароля

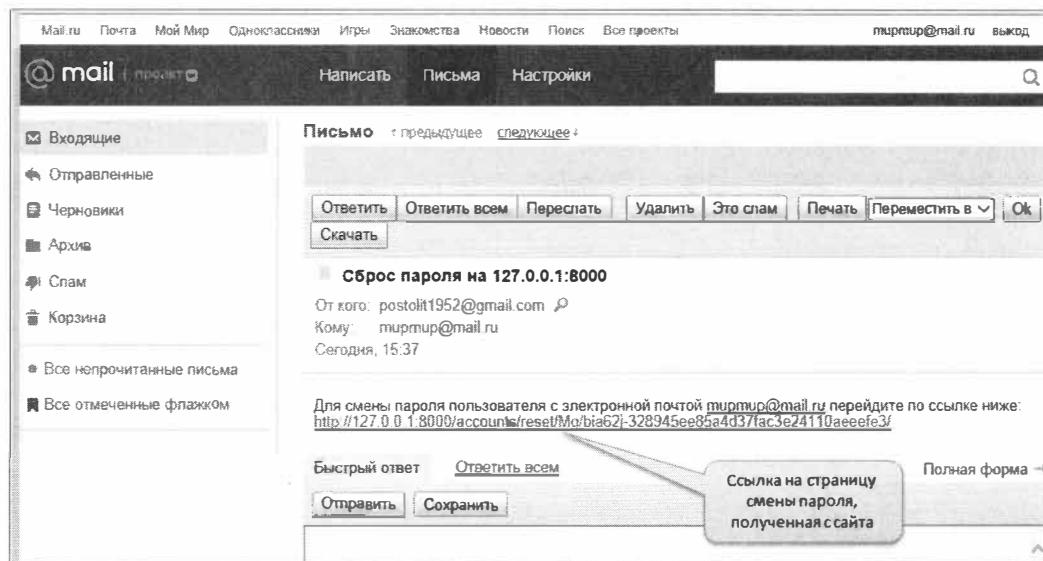


Рис. 11.36. Ссылка на страницу смены пароля на почте зарегистрированного пользователя системы

Если теперь щелкнуть левой кнопкой мыши на полученной ссылке, то пользователю будет загружена страница сайта, где он может поменять свой пароль (рис. 11.37).

#### **ПРИМЕЧАНИЕ**

Если вы будете использовать другие почтовые службы (mail.ru, yandex.ru и др.), то нужно применять настроочные параметры именно этих служб, с которыми можно ознакомиться на соответствующих сайтах.

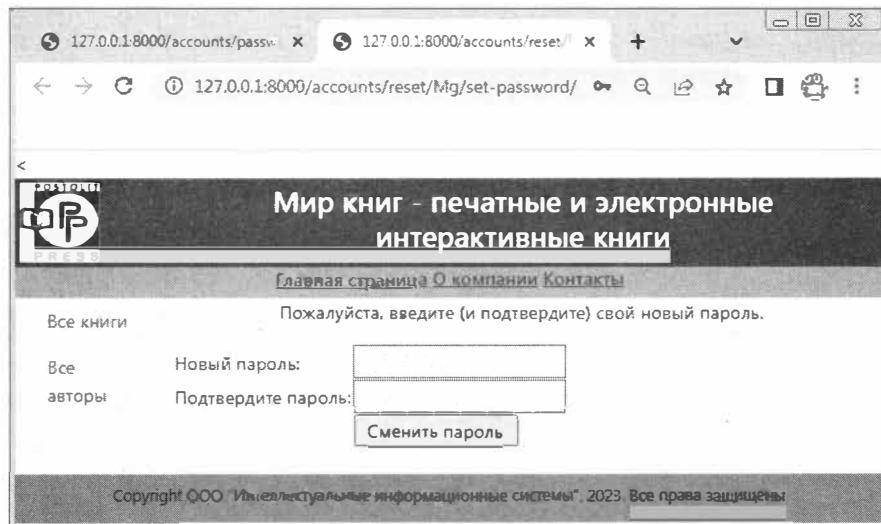


Рис. 11.37. Страница сайта для смены пароля

## 11.4. Проверка подлинности входа пользователя в систему

При желании можно получить и вывести на главную страницу сайта информацию о текущем статусе зарегистрированных в системе пользователей (вошел на сайт, покинул сайт). Для этого в шаблонах присутствует переменная {{user}}, которая добавляется в контекст шаблона по умолчанию.

Обычно при загрузке страницы сначала проверяется переменная шаблона {{user.is\_authenticated}}, позволяющая определить, имеет ли пользователь право просматривать конкретный контент. Чтобы обеспечить возможность входа (выхода) на сайт зарегистрированных пользователей, мы обновим нашу боковую панель главного шаблона (base.html). Отобразим на этой панели ссылку **Вход** (если пользователь не входил в систему или вышел из нее) и ссылку **Выход** (если он вошел в систему).

Откройте базовый шаблон \catalog\templates\base.html и добавьте код листинга 11.14 (изменения выделены серым фоном) в блок sidebar сразу после ссылки на страницу **Все авторы** (рис. 11.38).

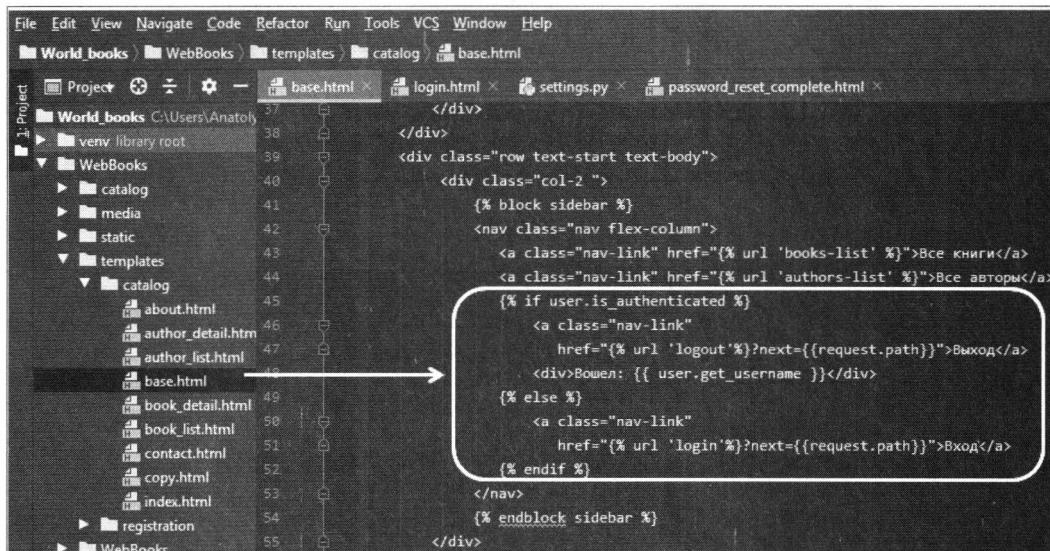
### Листинг 11.14. Измененный код файла \catalog\templates\base.html

```
{% block sidebar %}  
<nav class="nav flex-column">  
    <a class="nav-link" href="{% url 'books-list' %}">Все книги</a>  
    <a class="nav-link" href="{% url 'authors-list' %}">Все авторы</a>  
    {% if user.is_authenticated %}  
        <a class="nav-link"  
            href="{% url 'logout' %}?next={{request.path}}>Выход</a>  
    <div>Вошел: {{ user.get_username }}</div>
```

```

    {% else %}
        <a class="nav-link"
            href="{% url 'login' %}?next={{request.path}}>Вход</a>
    {% endif %}
</nav>
{% endblock sidebar %}

```



**Рис. 11.38.** Блок проверки входа/выхода пользователя на сайт в боковой панели базового шаблона

В этом коде используются теги шаблона `if...else...endif` для условного отображения текста в зависимости от того, является ли значение `{(user.is_authenticated)}` истинным. Если пользователь аутентифицирован, то мы знаем, что это вошел зарегистрированный пользователь, поэтому вызываем `{(user.get_username)}`, чтобы отобразить его имя.

Затем мы создаем URL-адрес для входа и выхода из системы, используя тег шаблона URL-адреса и имена соответствующих конфигураций `url`. Обратите также внимание на то, как мы добавили фрагмент `?next={{request.path}}` в конец блока `url`. Это означает, что следующим URL-адресом будет URL-адрес текущей страницы. После успешного входа пользователя в систему представления будут использовать значение `next`, чтобы при выходе из системы перенаправить пользователя обратно на страницу, с которой он входил в систему.

Протестируем то, что мы сейчас сделали. Если теперь загрузить главную страницу сайта, то мы увидим, что на боковой панели появилась дополнительная ссылка с надписью **Вход** (рис. 11.39).

Щелкните левой кнопкой мыши на ссылку **Вход**, и вы попадете на страницу идентификации пользователя (рис. 11.40).

Введите на этой странице имя и пароль нашего тестового пользователя и нажмите на ссылку **Вход** — мы опять попадем на главную страницу сайта, но при этом в боковой панели появится имя вошедшего пользователя, а ссылка **Вход** будет заменена на ссылку **Выход** (рис. 11.41).



Рис. 11.39. Ссылка Вход на главной странице сайта, ведущая на страницу идентификации пользователя

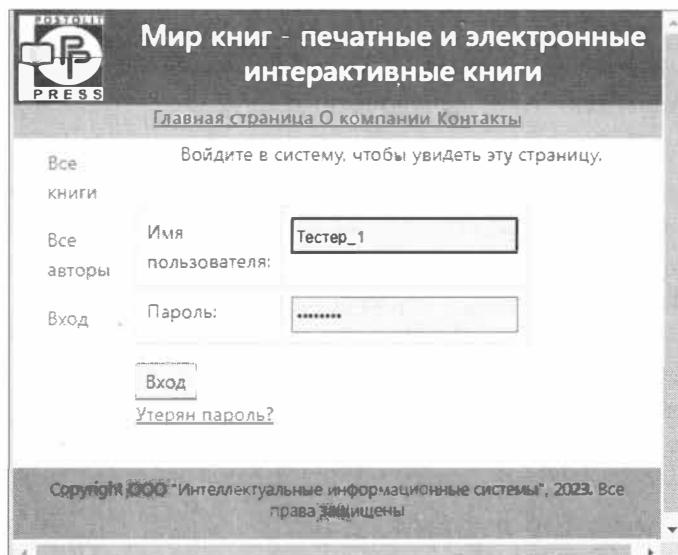


Рис. 11.40. Страница идентификации пользователя, на которую он попадает, нажав на ссылку Вход на главной странице сайта



Рис. 11.41. Главная страница сайта после успешной идентификации пользователя

При нажатии на ссылку **Выход** пользователь вернется на главную страницу сайта, при этом имя пользователя исчезнет, а ссылка **Выход** изменится на ссылку **Вход**.

В итоге мы реализовали возможность идентификации пользователей. Это обеспечивает разделение всех посетителей сайта на две категории:

- обычные пользователи — с ограничением доступа к некоторым страницам сайта;
- зарегистрированные пользователи — с возможностью входа на страницы сайта, которые недоступны для обычных пользователей.

Такое разделение достаточно часто встречается на сайтах различного типа:

- интернет-магазины (обычным посетителям доступен только просмотр каталога с товарами, а заказы могут делать только зарегистрированные пользователи);
- социальные сети (обычные посетители могут только ознакомиться с условиями размещения своих страниц, а зарегистрированные пользователи могут сформировать свои страницы, выкладывать на них информацию, общаться с другими пользователями);
- блоги (обычные посетители могут только просматривать страницы блога, а зарегистрированные пользователи — оставлять свои отзывы) и т. п.

## 11.5. Формирование страниц сайта для создания заказов на книги

До настоящего момента у пользователей не было возможности делать заказы на книги (через интернет-магазин или библиотеку). Давайте, реализуем такую возможность,

представив, что сайт «Мир книг» — это некий аналог электронной библиотеки для удаленных пользователей или площадки для продажи книг.

Начнем с того, что расширим модель BookInstance (экземпляры книг) так, чтобы получить возможность использования приложения Django Admin для оформления заказа (выдачи) книг нашему тестовому пользователю.

Прежде всего, мы должны предоставить пользователям возможность взять книгу на какое-то время. В модели BookInstance уже есть поле для обозначения статуса экземпляра книги (status) и поле для хранения даты возврата книги (due\_back), но у нас пока нет связи между этой моделью и моделью «пользователи». Мы создадим такую связь с помощью поля ForeignKey («один ко многим»), т. е. «один пользователь — несколько экземпляров книг». Кроме того, нужно будет создать простой механизм для проверки условия, не просрочил ли пользователь возврат полученной книги.

Импортируем модель User из модели django.contrib.auth.models, которая была создана в административной панели. Для этого откроем файл catalog\models.py и добавим строку кода (листинг 11.15) чуть ниже предыдущей строки импорта в верхней части файла (рис. 11.42).

#### Листинг 11.15. Измененный код файла \catalog\models.py

```
from django.contrib.auth.models import User
```

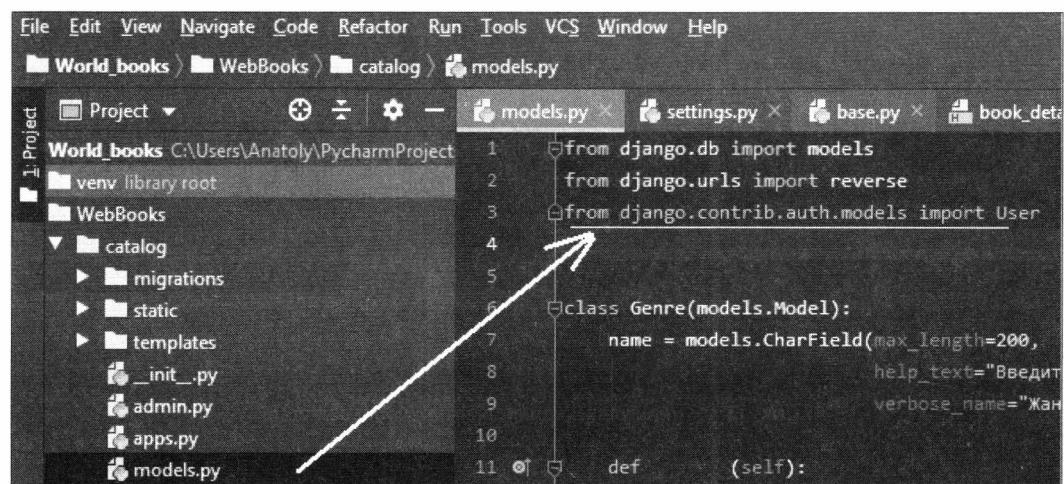


Рис. 11.42. Импортируем модель User из django.contrib.auth.models

Добавим в модель BookInstance поле borrower (заемщик, получатель) в классе BookInstance (листинг 11.16, добавленный код выделен серым фоном).

#### Листинг 11.16. Измененный код файла \catalog\models.py

```
# экземпляр книги
class BookInstance(models.Model):
    book = models.ForeignKey('Book', on_delete=models.CASCADE, null=True)
```

```

inv_nom = models.CharField(
    max_length=20,
    null=True,
    help_text="Введите инвентарный номер экземпляра",
    verbose_name="Инвентарный номер")
status = models.ForeignKey('Status',
    on_delete=models.CASCADE,
    null=True,
    help_text='Изменить состояние экземпляра',
    verbose_name="Статус экземпляра книги")
due_back = models.DateField(null=True,
    blank=True,
    help_text="Введите конец срока статуса",
    verbose_name="Дата окончания статуса")
borrower = models.ForeignKey(User, on_delete=models.SET_NULL,
    null=True, blank=True,
    verbose_name="Заказчик",
    help_text='Выберите заказчика книги')
objects = models.Manager
# Метаданные
class Meta:
    ordering = ["due_back"]
def __str__(self):
    return '%s %s %s' % (self.inv_nom, self.book, self.status)

```

Теперь добавим в эту модель свойство, которое в наших шаблонах позволит показать, просрочен ли конкретный экземпляр книги. Хотя мы могли бы сделать такую проверку в самом шаблоне, но использование свойства представляется более эффективным. Так что добавим в верхнюю часть файла `catalog\models.py` импорт модуля для обработки дат (листинг 11.17) и определение свойства внутри класса `BookInstance` (листинг 11.18).

#### Листинг 11.17. Измененный код файла `\catalog\models.py`

```
from datetime import date
```

#### Листинг 11.18. Измененный код файла `\catalog\models.py`

```

@property
def is_overdue(self):
    if self.due_back and date.today() > self.due_back:
        return True
    return False

```

Здесь проверяется условие — не превысило ли значение даты возврата книги текущей даты. В зависимости от значений этих дат будут возвращаться значения `True` или `False`.

Так как мы обновили наши модели, нужно изменить проект — создать файл миграции и внести соответствующие изменения в саму базу данных. Для этого в окне терминала PyCharm последовательно выполним следующие команды:

```
python manage.py makemigrations
python manage.py migrate
```

Теперь откроем файл catalog\admin.py и добавим поле borrower в класс BookInstanceAdmin — как в list\_display, так и в fieldsets. После этих изменений код для класса BookInstanceAdmin будет выглядеть как в листинге 11.19 (изменения выделены серым фоном).

#### Листинг 11.19. Измененный код файла catalog\admin.py

```
# Регистрируем класс BookInstanceAdmin для экземпляров книг
@admin.register(BookInstance)
class BookInstanceAdmin(admin.ModelAdmin):
    list_display = ('book', 'status', 'borrower', 'due_back', 'id')
    list_filter = ('book', 'status')
    fieldsets = (
        ('Экземпляр книги', {
            'fields': ('book', 'inv_nom')}),
        ('Статус и окончание его действия', {
            'fields': ('status', 'due_back', 'borrower')}),
    )
```

Эти изменения сделают поле borrower видимым в разделе Admin, так что мы сможем при необходимости назначить пользователя User для экземпляра книги BookInstance.

Теперь, когда можно оформить выдачу книги конкретному пользователю, зайдем в панель администратора и закрепим за нашим тестовым пользователем несколько экземпляров книг в разделе **Book Instances**: установим тестовому пользователю значение поля borrowed, присвоим значение status полю **В заказе** и назначим сроки возврата как на будущие, так и на прошедшие даты.

Итак, запускаем наш сайт, входим на страницу административной панели с паролем суперпользователя (см. рис. 8.31) и щелкнем левой кнопкой мыши на ссылке **Book Instances** — на открывшейся странице в таблице со списком экземпляров книг появилась колонка **Заказчик** (рис. 11.43).

Как можно видеть из данного рисунка, статус **В заказе** имеют две книги. Загрузим страницу с экземпляром книги «Дубровский», имеющей статус **В заказе**, — на открывшейся странице с информацией об этом экземпляре книги появилось новое поле: **Заказчик**. В выпадающем списке выберем для этой книги в качестве заказчика пользователя с именем **Тестер\_1** и сделаем этот заказ просроченным (выберем любую дату из прошедшего периода). В итоге получим следующую страницу (рис. 11.44).

При нажатии на кнопку **СОХРАНИТЬ** будет выполнен возврат на предыдущую страницу, где мы увидим внесенные нами изменения (рис. 11.45).

Проделаем ту же процедуру с другой книгой, имеющей статус **В заказе**, но установим для нее дату возврата из будущего периода (рис. 11.46).

Теперь создадим представление (view) для получения списка всех книг, которые были заказаны пользователем. На этот раз мы создадим представление на основе другого базового класса Django — LoginRequiredMixin. Представление (view), созданное на основе этого класса, сможет вызвать только пользователь, который вошел в систему со своим паролем. Мы также явно зададим имя шаблону (`template_name`) вместо имени шаблона по умолчанию. Это связано с тем, что у нас может быть несколько разных списков

Администрирование Django

Start typing to filter...

CATALOG

- Authors + Добавить
- Book instances + Добавить
- Books + Добавить
- Genres + Добавить
- Languages + Добавить
- Publishers + Добавить
- Statuses + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

- Группы + Добавить
- Пользователи + Добавить

Выберите book instance для изменения

Действие: Выполнить Выбрано 0 объектов из 9

	Файл	СТАТУС ЭКЗЕМПЛЯРА КНИГИ	ЗАКАЗЧИК	ДАТА ОКОНЧАНИЯ СТАТУСА	Ю
<input type="checkbox"/>	вок	На складе	-		8
<input type="checkbox"/>	Дубровский	Продана	-		7
<input type="checkbox"/>	Война и мир	На складе	-		6
<input type="checkbox"/>	Война и мир	На складе	-		5
<input type="checkbox"/>	Золотой теленок	В заказе	-		4
<input type="checkbox"/>	Золотой теленок	На складе	-		3
<input type="checkbox"/>	12 стульев	На складе	-		2
<input type="checkbox"/>	12 стульев	На складе	-		1
<input type="checkbox"/>	12 стульев	На складе	-	18 января 2023 г.	9

9 book instances

Рис. 11.43. Колонка «Заказчик» в списке экземпляров книг

Начало > Catalog > Book instances > 2 Дубровский В заказе

Start typing to filter...

CATALOG

- Authors + Добавить
- Book instances + Добавить
- Books + Добавить
- Genres + Добавить
- Languages + Добавить
- Publishers + Добавить
- Statuses + Добавить

ПОЛЬЗОВАТЕЛИ И ГРУППЫ

- Группы + Добавить
- Пользователи + Добавить

Изменить book instance

2 Дубровский В заказе

История

Экземпляр книги

Book: Дубровский

Инвентарный номер:

2

Введите инвентарный номер экземпляра

Статус и окончание его действия

Статус экземпляра книга:

В заказе

Изменить состояние экземпляра

Дата окончания статуса:

12.01.2023

Сегодня

Введите конец срока статуса

Внимание: Ваше локальное время отстает от времени сервера на 3 часа.

Заказчик:

Тестер\_1

Выберите заказчика книги

Измененные параметры

Удалить Сохранить и добавить другой объект Сохранить

Рис. 11.44. Выбор заказчика на конкретный экземпляр книги

Выберите book instance для изменения

Действие: Выполнить Выбрано 0 объектов из 9

<input type="checkbox"/>	BOOK	СТАТУС ЭКЗЕМПЛЯРА КНИГИ	ЗАКАЗЧИК	ДАТА ОКОНЧАНИЯ СТАТУСА	ID
<input type="checkbox"/>	Дубровский	На складе	-	-	8
<input type="checkbox"/>	Война и мир	Продана	-	-	7
<input type="checkbox"/>	Война и мир	На складе	-	-	6
<input type="checkbox"/>	Золотой теленок	В заказе	-	-	5
<input type="checkbox"/>	Золотой теленок	На складе	-	-	4
<input type="checkbox"/>	12 стульев	На складе	-	-	3
<input type="checkbox"/>	12 стульев	На складе	-	-	2
<input type="checkbox"/>	12 стульев	На складе	-	-	1
<input type="checkbox"/>	Дубровский	В заказе	Тестер_1	12 января 2023 г.	9

9 book instances

Рис. 11.45. Результаты изменения поля ЗАКАЗЧИК и даты окончания статуса экземпляр книги

<input type="checkbox"/>	12 стульев	На складе	-	Измененные параметры
<input type="checkbox"/>	12 стульев	На складе	-	
<input type="checkbox"/>	12 стульев	На складе	-	
<input type="checkbox"/>	Дубровский	В заказе	Тестер_1	12 января 2023 г.
<input type="checkbox"/>	Золотой теленок	В заказе	Тестер_1	9 февраля 2023 г.

Рис. 11.46. Две книги со статусом просроченного и непросроченного заказа

BookInstance (с разными представлениями и разными шаблонами). Реализуем намеченные изменения, добавив в файл catalog\views.py код листинга 11.20.

#### Листинг 11.20. Измененный код файла \catalog\views.py

```
from django.contrib.auth.mixins import LoginRequiredMixin
from django.views import generic
class LoanedBooksByUserListView(LoginRequiredMixin, generic.ListView):
    # Универсальный класс представления списка книг,
    # находящихся в заказе у текущего пользователя
    model = BookInstance
    template_name ='catalog/bookinstance_list_borrowed_user.html'
    paginate_by = 10
    def get_queryset(self):
        return BookInstance.objects.filter(
            borrower=self.request.user).filter(
            status__exact='2').order_by('due_back')
```

Чтобы получить только объекты BookInstance для текущего пользователя, мы реализуем здесь собственный метод `get_queryset()` (заказы пользователя). Обратите внимание, что параметр '2' в этом фильтре — это код справочника статусов **В заказе**. Здесь мы сортируем книги по дате `due_back`, чтобы сначала отображались самые «старые» элементы из списка заказанных книг.

Затем откроем файл `\catalog\urls.py` и добавим адрес `url()`, который укажет на созданное представление — `LoanedBooksByUserListView`. Добавленные строки выделены в листинге 11.21 серым фоном.

#### Листинг 11.21. Измененный код файла `\catalog\urls.py`

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),
         name='authors-detail'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('mybooks/', views.LoanedBooksByUserListView.as_view(),
         name='my-borrowed'),
]
```

Теперь нам нужно создать шаблон для страницы, на которой будут показаны книги со статусом **В заказе**. Сначала создайте файл для этого шаблона:

`\templates\catalog\bookinstance_list_borrowed_user.html`

Затем напишите в нем код листинга 11.22.

#### Листинг 11.22. Код файла `\templates\catalog\bookinstance_list_borrowed_user.html`

```
{% extends "catalog/base.html" %}
{% block content %}
    <h1>Взятые книги</h1>
    {% if bookinstance_list %}
        {% for bookinst in bookinstance_list %}
            <div class="{% if bookinst.is_overdue %} text-danger{% endif %}">
                <a href="{% url 'book-detail' bookinst.book_pk %}">
                    {{bookinst.book.title}}</a>
                    {{ bookinst.due_back }}
            </div>
        {% endfor %}
    {% else %}
        <p>Нет книг со статусом В ЗАКАЗЕ.</p>
    {% endif %}
{% endblock %}
```

Этот шаблон очень похож на те, которые мы создали ранее для объектов книги (Book) и авторы (Author). Здесь сначала проверяется условие — есть ли заказы у данного пользователя, т. е. не пустой ли список заказов `{% if bookinstance_list %}`. Если список заказов пуст, то выводится сообщение «Нет книг со статусом В ЗАКАЗЕ». Если у пользователя есть заказы, то организуется цикл `for`, в котором выдается информация о заказанных книгах — `{% for bookinst in bookinstance_list %}`. Обратите внимание, что в теле цикла есть инструкция `if` для проверки условия, не просрочен ли заказ — `{% if bookinst.is_overdue %}`. Если заказ окажется просроченным, то информация о нем (дата возврата книги) будет показана красным цветом (`text-danger`). А название книги, которая находится в заказе, представляет собой ссылку на страницу с информацией об этой книге:

```
href="{% url 'book-detail' bookinst.book %}"
```

Для установления факта, что заказ просрочен, здесь предусмотрен метод `is_overdue`, который создан в модели экземпляров книг:

```
{% if bookinst.is_overdue %}
```

Так как мы создали новую HTML-страницу с заказами пользователя, то создадим ссылку на нее. Откроем главный шаблон `base.html` и в блоке `sidebar` добавим код листинга 11.23 (изменения выделены серым фоном).

#### Листинг 11.23. Измененный код файла `\templates\catalog\base.html`

```
<div class="col-2">
    {% block sidebar %}
    <nav class="nav flex-column">
        <a class="nav-link" href="{% url 'books-list' %}">Все книги</a>
        <a class="nav-link" href="{% url 'authors-list' %}">Все авторы</a>
        {% if user.is_authenticated %}
            <a class="nav-link"
                href="{% url 'logout' %}?next={{request.path}}">Выход</a>
            <div>Бошел: {{ user.get_username }}</div>
            <a class="nav-link"
                href="{% url 'my-borrowed' %}">Мои заказы</a>
        {% else %}
            <a class="nav-link"
                href="{% url 'login' %}?next={{request.path}}">Вход</a>
        {% endif %}
    </nav>
    {% endblock sidebar %}
</div>
```

Здесь после строки, в которой выводится имя зарегистрированного пользователя, добавлена ссылка на страницу с заказами этого пользователя:

```
<a class="nav-link" href="{% url 'my-borrowed' %}">Мои заказы</a>
```

Протестируем работу сайта после внесенных изменений. После запуска сервера открывается главная страница сайта (рис. 11.47).

Пока на ней нет ничего необычного — ведь мы еще не прошли авторизацию в качестве зарегистрированного пользователя. Нажмите на ссылку **Вход** и пройдите авторизацию как пользователь с именем **Тестер\_1** (рис. 11.48).

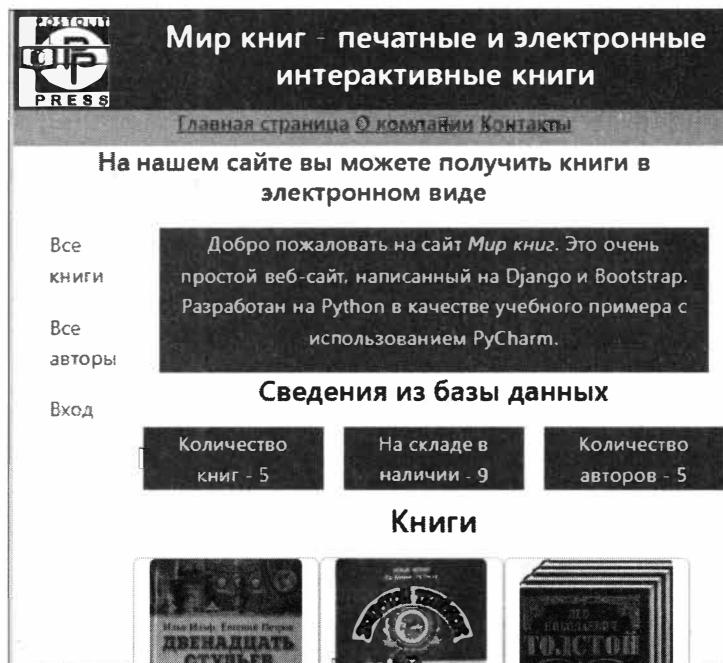


Рис. 11.47. Главная страница сайта «Мир книг» с возможностью входа зарегистрированных пользователей

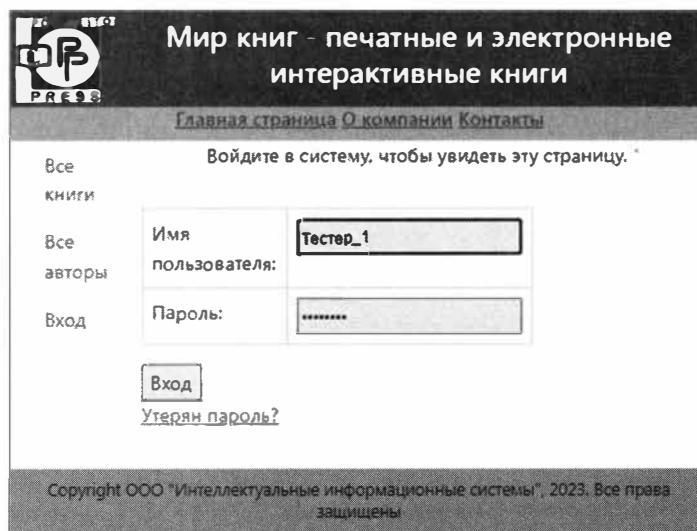


Рис. 11.48. Страница сайта «Мир книг» для входа зарегистрированных пользователей

Если после авторизации снова нажать на ссылку **Вход**, то вы вернетесь на главную страницу сайта, но уже с возможностями зарегистрированного пользователя (рис. 11.49). На этой странице мы видим как минимум два признака того, что пользователь прошел авторизацию: появилось имя пользователя (**Тестер\_1**) и появилась ссылка на страницу

**Мои заказы** для этого пользователя. Если теперь щелкнуть левой кнопкой мыши на ссылке **Мои заказы**, то откроется новая страница с книгами, которые заказал именно этот пользователь (рис. 11.50). Причем книга с просроченным сроком возврата будет выделена красным цветом.



Рис. 11.49. Страница сайта «Мир книг» после входа авторизованного пользователя

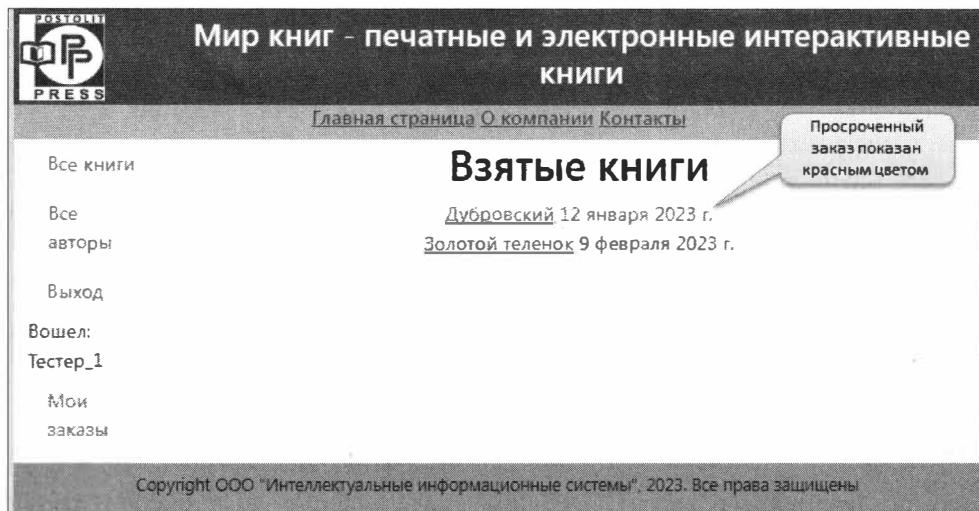


Рис. 11.50. Страница сайта с перечнем книг со статусом В заказе пользователя Тестер\_1

Если на этой странице нажать кнопку **Выход**, а потом перейти на главную страницу, то ссылка **Мои заказы** исчезнет, т. к. в системе будет находиться уже неавторизованный пользователь.

## 11.6. Работа с формами

В этом разделе мы рассмотрим возможности работы с HTML-формами и, в частности, познакомимся с самым простым способом построения формы для создания, обновления и удаления экземпляров модели. При этом мы расширим функциональность сайта «Мир книг», чтобы появилась возможность добавлять и редактировать книги и их авторов, используя собственные формы, а не средства административной панели.

*Форма HTML* — это группа из одного или нескольких полей (вернее, их виджетов) на веб-странице, которая служит для ввода данных со стороны пользователей для последующей отправки их на сервер. Формы являются гибким механизмом сбора пользовательских данных, поскольку имеют целый набор виджетов для ввода различных типов данных: текстовые поля, флажки, переключатели, установщики дат, изображения, файлы и т. п.

Формы предоставляют безопасный способ взаимодействия пользовательского клиента и сервера, поскольку позволяют отправлять данные в POST-запросах, обеспечивая защиту от межсайтовой подделки запроса (Cross Site Request Forgery, CSRF).

Пока мы не создавали формы для нашего сайта, но встречались с ними в административной панели Django, например применяли встроенную форму, состоящую из нескольких списков выбора и текстовых полей, для редактирования сведений о книгах.

Работа с формами может быть достаточно сложной. Разработчикам нужно описать форму на языке HTML и проверить ее правильность. Необходимо также проверять на стороне сервера введенные пользователем данные (а возможно, и на стороне клиента). При возникновении ошибок нужно снова показать пользователю форму и при этом указать на то, что пошло не так. В случае же успеха внести изменения в БД и проинформировать об этом пользователя. Приятно сознавать, что Django берет большую часть этой работы на себя. Данный фреймворк позволяет создать форму и ее поля в программе в виде объектов, а затем использовать эти объекты и для генерации кода HTML-формы, и для управления процессом корректного взаимодействия пользователя с формой.

В этом разделе вы познакомитесь с несколькими способами создания форм и работы с ними. В том числе вы узнаете, как применение классов в представлениях (*view*) для взаимодействия с формой может значительно уменьшить объем работы по написанию программного кода. Кроме того, мы дополним возможности нашего сайта «Мир книг», добавив в него страницы для ввода, редактирования, удаления книг и авторов, а также расширим стандартные возможности административной части сайта.

### 11.6.1. Краткий обзор форм в Django

Рассмотрим простейшую форму HTML, имеющую поле для ввода имени и связанную с этим полем текстовую метку (рис. 11.51).

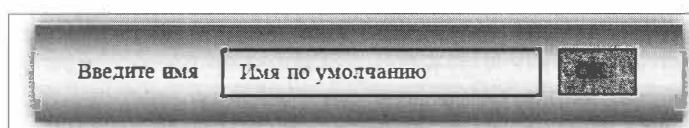


Рис. 11.51. Простейшая форма

Форма описывается на языке HTML в виде набора элементов, расположенных внутри парных тегов `<form>...</form>`. Любая форма содержит как минимум одно поле-тег `input` типа `type="submit"`. Например, форма, изображенная на рис. 11.51, будет создана на основе кода листинга 11.24.

#### Листинг 11.24. Пример кода формы Django

```
<form action="/team_name_url/" method="post">
    <label for="team_name">Введите имя</label>
    <input id="team_name" type="text" name="name_field"
           value="Имя по умолчанию">
    <input type="submit" value="OK">
</form>
```

Здесь у нас только одно поле для ввода имени, но форма может иметь любое количество элементов ввода и связанных с ними текстовых меток. Атрибут элемента `type` определяет, виджет какого типа будет показан в той или иной строке. Атрибуты `name` и `id` служат для однозначной идентификации поля в коде JavaScript, CSS и HTML, в то время как `value` содержит значение поля, когда оно показывается в первый раз. Текстовая метка добавляется при помощи тега `label` (строка с текстом Введите имя в листинге 11.24) и имеет атрибут `for` со значением идентификатора `id` того поля, с которым связана эта текстовая метка.

Элемент `input` типа `type="submit"` будет по умолчанию показан как кнопка **OK**, нажав на которую пользователь отправляет введенные им данные на сервер (в нашем случае — только значение поля с идентификатором `team_name`). Атрибуты формы `action` и `method` определяют, каким методом будут отправлены данные на сервер (атрибут `method`) и куда (атрибут `action`).

Рассмотрим более подробно два последних элемента:

- `action` — это ресурс (URL-адрес), куда будут отправлены данные для обработки. Если значение не установлено (т. е. значением поля является пустая строка), тогда данные будут отправлены в представление `view` (функцию или класс), которое сформировало текущую страницу;
- `method` — определяет метод отправки данных (POST или GET):
  - метод POST должен использоваться тогда, когда необходимо внести изменения в базу данных на сервере. Применение этого метода должно повысить уровень защиты от межсайтовой подделки запроса (CSRF);
  - метод GET предназначен только для форм, действия с которыми не приводят к изменению базы данных (например, для поисковых запросов). Кроме того, этот метод рекомендуется применять для создания внешних ссылок на ресурсы сайта.

Роль сервера в первую очередь заключается в выводе начального состояния формы (либо содержащей пустые поля, либо с установленными начальными значениями). После того как пользователь нажмет на кнопку **OK**, сервер получит все данные формы и должен провести их проверку на корректность. В том случае, если форма содержит неверные данные, сервер должен снова отобразить пользователю форму, показав при этом поля с правильными данными, а также сообщения о том, что нужно исправить

в полях с ошибочными данными. В тот момент, когда сервер получит запрос с правильными данными, он должен выполнить все необходимые действия (например, сохранить данные в БД, вернуть результат поиска, загрузить файлы и т. п.) и при необходимости проинформировать пользователя об успешном завершении операции.

Как видите, создание HTML-формы, проверка и возврат данных, перерисовка введенных значений, а также выполнение желаемых действий с правильными данными может потребовать весьма больших усилий, чтобы все это «заработало». Django делает этот процесс намного проще, беря на себя создание некоторых «тяжелых» и повторяющихся фрагментов кода.

## 11.6.2. Управление формами в Django

Для управления формами Django задействует тот же механизм, который мы изучали в предыдущих разделах — представление (view):

- получает запрос от пользователя и выполняет необходимые действия, включающие чтение данных из БД;
- генерирует и возвращает пользователю HTML-страницу из шаблона, в который передаются полученные из БД данные.

При использовании форм на серверной части системы нужно дополнительно обработать данные, предоставленные пользователем, и в случае возникновения ошибок снова перерисовать исходную страницу. Схема, приведенная на рис. 11.52, демонстрирует процесс взаимодействия пользователя с формой в Django.

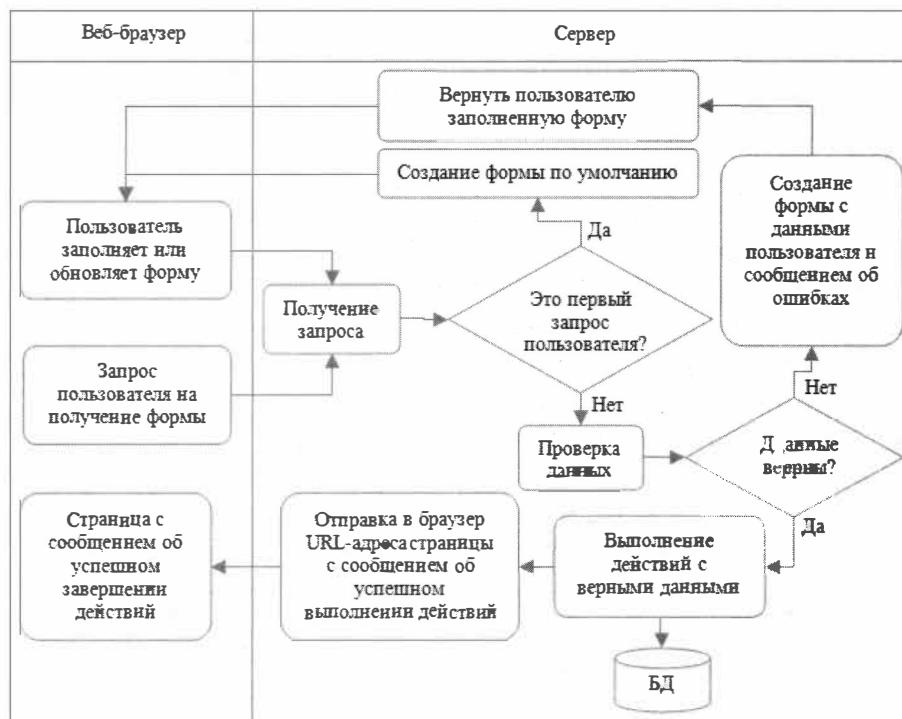


Рис. 11.52. Процесс взаимодействия пользователя с формой Django

В соответствии с этой схемой главными действиями, которые берут на себя формы Django, являются:

- показ формы по умолчанию при первом запросе со стороны пользователя:
  - форма может содержать пустые поля (например, если вы создаете новую запись в базе данных), или эти поля могут иметь начальные значения (например, если вы изменяете запись или хотите заполнить ее каким-либо начальным значением);
  - форма в начальный момент является несвязанной, потому что не ассоциируется с какими-либо введенными пользователем данными (хотя и может иметь начальные значения);
- получение данных из формы со стороны клиента и связывание их с формой (классом формы) на стороне сервера:
  - связывание данных с формой означает, что данные, введенные пользователем, а также возможные ошибки при ее обновлении будут относиться именно к этой форме, а не к какой-либо иной;
- очистка и проверка данных:
  - очистка данных — это их проверка на наличие недопустимых символов или вставок в поля ввода (с удалением символов, которые потенциально могут привести к отправке вредоносного содержимого на сервер) с последующей конвертацией очищенных данных в подходящие типы данных Python;
  - при проверке данных анализируются значения полей (например, правильность введенных дат, их диапазон и т. п.);
- если какие-либо данные неверны, то выполняется обновление формы, но на этот раз с уже введенными пользователем данными и сообщениями об ошибках;
- если все данные верны, то происходит выполнение необходимых действий (например, сохранение данных в БД, отправка писем, возврат результата поиска, загрузка файла и т. д.);
- когда все действия успешно завершены, происходит перенаправление пользователя на другую страницу.

Django предоставляет несколько инструментов и приемов, которые помогают пользователю во время выполнения этих задач. Наиболее важным из этих инструментов является класс `Form`, который упрощает генерацию HTML-формы, а также очистку и проверку данных. В следующем разделе мы рассмотрим процесс работы с формами на основе практического примера по созданию страницы, которая позволит пользователям сайта обновлять информацию об авторах и книгах. При этом будут использованы два базовых класса Django: `Form` и `ModelForm`.

### 11.6.3. Форма для ввода и обновления информации об авторах книг на основе класса `Form()`

Для начала создадим представление (`view`) и шаблон HTML-страницы, на которую выведем список авторов. Это будет таблица, в которой мы создадим ссылки на формы для выполнения следующих действий: добавить нового автора, удалить автора, изменить

сведения об авторе. Здесь нам пока еще не потребуются формы Django, а сведения об авторах получим из модели.

Для начала создадим представление (*view*), которое будет обрабатывать запросы пользователей. Откроем файл `WebBooks/catalog/views.py` и напишем в нем код листинга 11.25.

### Листинг 11.25. Измененный код файла `WebBooks/catalog/views.py`

```
# вызов страницы для редактирования авторов
def edit_authors(request):
    author = Author.objects.all()
    context = {'author': author}
    return render(request, "catalog/edit_authors.html", context)
```

Это очень простое представление. Здесь создается объект `author`, в который из БД загружаются сведения обо всех авторах. Этот объект упаковывается в словарь `context`, который затем передается в шаблон `edit_authors.html`.

Создадим теперь в каталоге `World_Books\WebBooks\templates\catalog` файл с именем `edit_authors.html` и напишем в нем код листинга 11.26.

### Листинг 11.26. Код файла `World_Books\WebBooks\templates\catalog\edit_authors.html`

```
{% extends "catalog/base.html" %}
{% block content %}


##### Список авторов для редактирования



Добавить автора



| Id | Имя | Фамилия | Редактировать |
|----|-----|---------|---------------|
|----|-----|---------|---------------|


```

```
{% endfor %}  
{% else %}  
    <p> В базе данных нет авторов</p>  
{% endif %}  
    <tbody>  
    </table>  
    </div>  
    <div class="col-2"></div>  
    </div>  
{% endblock %}
```

В первой строке выводится заголовок страницы, а во второй строке находится ссылка на страницу с добавлением нового автора:

```
<a href="/authors_add/{{ authors_add }}">Добавить автора</a>
```

Затем проверяется условие, есть ли в БД записи с информацией об авторах с использованием инструкции if — {% if author %}. Если информация об авторах отсутствует, то в форме будет выдано сообщение «В базе данных нет авторов».

Если в БД есть записи об авторах, то далее создается таблица с четырьмя колонками:

- идентификатор записи в БД;
- имя автора;
- фамилия автора;
- колонка со ссылками на опции редактирования (**Изменить, Удалить**).

Формирование строк таблицы происходит в цикле, созданном на основе инструкции for — {% for author in author.all %}.

Теперь для доступа к данному шаблону нужно прописать маршрут. Открываем модуль World\_Books\WebBooks\catalog\urls.py и добавляем в него строку из листинга 11.27 (изменения выделены серым фоном).

#### Листинг 11.27. Измененный код файла World\_Books\WebBooks\catalog\urls.py

```
from django.urls import path  
from .import views  
urlpatterns = [  
    path('', views.index, name='index'),  
    path('books/', views.BookListView.as_view(), name='books-list'),  
    path('books/<int:pk>/', views.BookDetailView.as_view(),  
         name='book-detail'),  
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),  
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),  
         name='authors-detail'),  
    path('about/', views.about, name='about'),  
    path('contact/', views.contact, name='contact'),  
    path('mybooks/', views.LoanedBooksByUserListView.as_view(),  
         name='my-borrowed'),  
    path('edit_authors/', views.edit_authors, name='edit_authors'),]
```

Остался последний шаг, в базовом шаблоне нужно добавить ссылку на страницу `edit_authors`. Открываем шаблон `base.html` и добавляем в него строку из листинга 11.28 (изменения выделены серым фоном).

#### Листинг 11.28. Измененный код файла `\WebBooks\templates\catalog\base.html`

```
<nav class="nav flex-column">
    <a class="nav-link" href="{% url 'books-list' %}">Все книги</a>
    <a class="nav-link" href="{% url 'authors-list' %}">Все авторы</a>
    {% if user.is_authenticated %}
        <a class="nav-link" href="{% url 'logout'%}?next={{request.path}}>Выход</a>
        <div>Вошел: {{ user.get_username }}</div>
        <a class="nav-link" href="{% url 'edit_authors' %}">Редактор авторов</a>
        <a class="nav-link" href="{% url 'my-borrowed' %}">Мои заказы</a>
    {% else %}
        <a class="nav-link" href="{% url 'login'%}?next={{request.path}}>Вход</a>
    {% endif %}
</nav>
{% endblock sidebar %}
```

Эта ссылка добавлена в боковую панель базового шаблона в блок, к которому имеют доступ только зарегистрированные пользователи. Таким образом, ссылка на страницу с редактированием сведений об авторах откроется только тогда, когда зарегистрированный пользователь войдет в систему.

Протестируем работу сайта после внесенных изменений. При запуске сервера открывается главная страница сайта. После регистрации пользователя в системе в меню боковой панели появится ссылка на страницу для редактирования данных об авторах (рис. 11.53).

Если теперь щелкнуть левой кнопкой мыши по данной ссылке, то откроется страница со списком авторов (рис. 11.54).

Как видно из данного рисунка, перед шапкой таблицы находится ссылка **Добавить автора**, а в каждой строке две ссылки: **Изменить** — изменить (отредактировать) сведения об авторе, **Удалить** — удалить данные об авторе из БД. Пока эти ссылки не работают, т. к. не созданы соответствующие страницы. Создадим формы, с помощью которых зарегистрированные пользователи смогут добавлять, редактировать и удалять сведения об авторах книг, и начнем с формы для страницы **Добавить автора**.

Поскольку каждая форма определяется в виде отдельного класса, который расширяет базовый класс `forms.Form`, то создадим в каталоге `World_Books\WebBooks\catalog` файл с именем `forms.py` (рис. 11.55).

В этом модуле напишем код формы для добавления в БД новых авторов (листинг 11.29).

The screenshot shows a user interface for managing books and authors. On the left, there's a sidebar with links like 'All books', 'All authors', 'Logout', 'Entered: Tester\_1', 'Editor of authors', and 'My orders'. The main content area has a heading 'Mir knig - printed and electronic interactive books'. Below it, a message says: 'Welcome to the Mir knig site. This is a very simple web site, built on Django and Bootstrap. Developed in Python using PyCharm.' A tooltip points to the 'Edit data from the database' link. At the bottom, there's a section for books ('Books - 5', 'In stock - 9', 'Count of authors - 5') and a preview of three book covers.

Рис. 11.53. Ссылка на страницу редактирования данных об авторах

The screenshot shows a list of authors for editing. The top navigation bar includes the logo 'POSTOLET PRESS' and links for 'Main page', 'About company', 'Contacts'. The main title is 'List of authors for editing'. Below it, a button 'Add author' is visible. The table lists five authors:

ID	Name	Surname	Action
1	Alexander	Belyayev	<a href="#">Edit</a> , <a href="#">Delete</a>
2	Alexander	Pushkin	<a href="#">Edit</a> , <a href="#">Delete</a>
3	Lev	Tolstoy	<a href="#">Edit</a> , <a href="#">Delete</a>
4	Ilya	Ilyf	<a href="#">Edit</a> , <a href="#">Delete</a>
5	Evgueniy	Petrov	<a href="#">Edit</a> , <a href="#">Delete</a>

At the bottom, a copyright notice reads: 'Copyright ООО "Интеллектуальные информационные системы", 2023. All rights reserved.'

Рис. 11.54. Страница со ссылками для редактирования данных об авторах

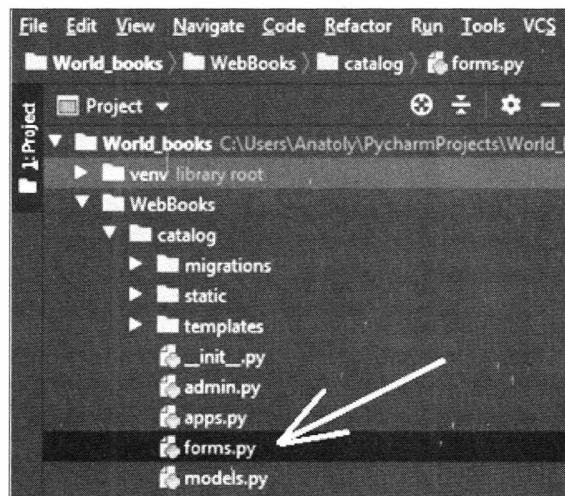


Рис. 11.55. Создание файла forms.py

**Листинг 11.29. Код файла World\_Books\WebBooks\catalog\forms.py**

```
from django import forms
from datetime import date
# форма для добавления в БД новых авторов
class Form_add_author(forms.Form):
    first_name = forms.CharField(label="Имя автора")
    last_name = forms.CharField(label="Фамилия автора")
    date_of_birth = forms.DateField(
        label="Дата рождения",
        initial=format(date.today()),
        widget=forms.widgets.DateInput(attrs={'type': 'date'}))
    about = forms.CharField(label="Сведения об авторе",
                           widget=forms.Textarea)
    photo = forms.ImageField(label="Фото автора")
```

Здесь мы создаем класс формы для добавления новых авторов — `Form_add_author` на основе базового класса `forms.Form`. Так как эта форма не связана с моделью данных, то поля для этой формы нужно определить самостоятельно. В данной форме заданы следующие поля для ввода данных:

- `first_name` — имя автора, имеет тип `forms.CharField` и будет генерировать на HTML-странице текстовое поле `input type="text"`;
- `last_name` — фамилия автора, имеет тип `forms.CharField` и будет генерировать на HTML-странице текстовое поле `input type="text"`;
- `date_of_birth` — дата рождения, имеет тип `forms.DateField` и будет генерировать поле для работы с датами `input type="date"`. Чтобы дата при первой загрузке формы не была пустой, мы инициировали для нее начальное значение — текущая дата `date.today()`. Кроме того, для данного поля сменили виджет — теперь это будет календарь, в котором можно выбрать нужную дату.

- `about` — сведения об авторе, имеет тип `forms.CharField`. Так как это поле предназначено для ввода многостраничного текста, то для него сменили виджет (`widget=forms.Textarea`);
- `photo` — поле для ввода фото (портрета) автора. Данное поле предназначено для ввода изображений, поэтому оно имеет тип `forms.ImageField`.

Затем в файл `views.py` добавим следующий код для вызова страницы, где пользователь сможет добавить нового автора, — это будет функция с именем `authors_add()` (листинг 11.30).

#### Листинг 11.30. Измененный код файла `\WebBooks\catalog\views.py`

```
# импорт для добавления автора
from .forms import Form_add_author
from django.urls import reverse
# Создание нового автора в БД
def add_author(request):
    if request.method == 'POST':
        form = Form_add_author(request.POST, request.FILES)
        if form.is_valid():
            # получить данные из формы
            first_name = form.cleaned_data.get("first_name")
            last_name = form.cleaned_data.get("last_name")
            date_of_birth = form.cleaned_data.get("date_of_birth")
            about = form.cleaned_data.get("about")
            photo = form.cleaned_data.get("photo")
            # создать объект для записи в БД
            obj = Author.objects.create(
                first_name=first_name,
                last_name=last_name,
                date_of_birth=date_of_birth,
                about=about,
                photo=photo)
            # сохранить полученные данные
            obj.save()
            # загрузить страницу со списком автором
            return HttpResponseRedirect(reverse('authors-list'))
    else:
        form = Form_add_author()
        context = {"form": form}
        return render(request, "catalog/authors_add.html", context)
```

Здесь сначала импортируется форма `Form_add_author` и функция Django `reverse`, а затем проверяется условие, какой запрос поступил от пользователя POST или GET.

Если поступил GET-запрос (пользователь открывает форму для ввода данных), то на основе класса `Form_add_author()` создается объект `form`, который через контекстный словарь `context` передается в шаблон `catalog/authors_add.html`.

Если поступил POST-запрос (пользователь ввел данные и нажал кнопку **Сохранить**), то сначала переменные (`first_name`, `last_name`, `date_of_birth`, `about`, `photo`) получают значе-

ния, которые пользователь ввел в поля формы, а затем на их основе создается специальный объект (`obj`), который записывается в БД (`obj.save`). После этого с помощью функции `reverse` вызывается страница со списком авторов (`authors-list`).

#### ПРИМЕЧАНИЕ

Обратите внимание на строку, в которой создается объект `form`: `form = Form_add_author(request.POST, request.FILES)`. Так как в форме вводится изображение, то здесь обязательно должен присутствовать параметр `request.FILES`. При его отсутствии файл с изображением не будет передаваться и не будет записан в папку `media/images/`.

Создадим теперь в папке `World_Books\WebBooks\templates\catalog` файл `authors_add.html`. (рис. 11.56). Это будет шаблон страницы для отображения формы ввода сведений о новом авторе, напишем в данном модуле код листинга 11.31.

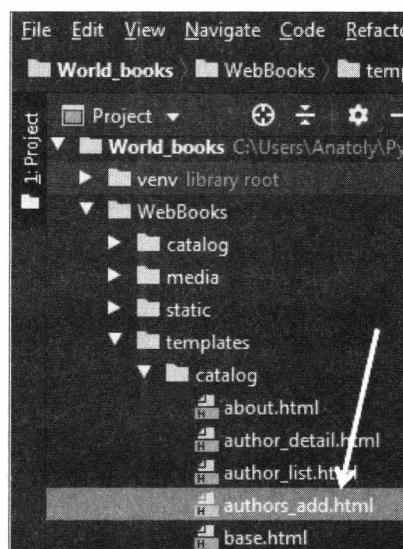


Рис. 11.56. Создание файла `authors_add.html`

#### Листинг 11.31. Код файла `World_Books\WebBooks\templates\catalog\authors_add.html`

```
% extends "catalog/base.html"
% block content
<h5>Добавить в БД автора книги</h5>
<form action="" method="POST" enctype="multipart/form-data">
    % csrf_token
    <div class="row">
        <div class="col-2 "></div>
        <div class="col-8 ">
            <table class="table text-start">
                {{ form }}
            </table>
        </div>
        <div class="col-2"></div>
    </div>
```

```
<input type="submit" value="Сохранить" >  
</form>  
{% endblock %}
```

В первой строке этого кода мы подключили базовый шаблон `base.html`. Затем в блоке `content` создана таблица, в которую и выводится наша форма — `{{ form }}`.

### ПРИМЕЧАНИЕ

В данном шаблоне обратите внимание на тег, в котором создается форма `<form:action="" method="POST" enctype="multipart/form-data">`. Так как в форме вводится изображение, то здесь обязательно должен присутствовать параметр `enctype="multipart/form-data"`. При его отсутствии файл с изображением не будет передаваться и не будет записан в папку `media/images/`.

Создавать ссылку для вызова этого шаблона не нужно, т. к. мы это сделали ранее в шаблоне `edit_authors.html`. Если вернуться к фрагменту кода этой страницы, то в приведенном далее фрагменте кода строка с соответствующей ссылкой выделена серым фоном:

```
{% extends "catalog/base.html" %}  
{% block content %}  
<div class="row text-start lh-2">  
    <h5>Список авторов для редактирования</h5>  
</div>  
    <div class="row">  
        <div class="col-2 "></div>  
            <a href="/authors_add/{{ authors_add }}"/>Добавить автора</a>  
        <div class="col-8 ">  
            <table class="table table-striped table-bordered text-start">
```

А вот создать маршрут для вызова шаблона `authors_add.html` необходимо. Для этого открываем модуль `World_Books\WebBooks\catalog\urls.py` и добавляем в него строку из листинга 11.32 (изменения выделены серым фоном).

#### Листинг 11.32. Измененный код файла `World_Books\WebBooks\catalog\urls.py`

```
from django.urls import path  
from . import views  
urlpatterns = [  
    path('', views.index, name='index'),  
    path('books/', views.BookListView.as_view(), name='books-list'),  
    path('books/<int:pk>/', views.BookDetailView.as_view(),  
         name='book-detail'),  
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),  
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),  
         name='authors-detail'),  
    path('about/', views.about, name='about'),  
    path('contact/', views.contact, name='contact'),  
    path('mybooks/', views.LoanedBooksByUserListView.as_view(),  
         name='my-borrowed'),  
    path('edit_authors/', views.edit_authors, name='edit_authors'),  
    path('authors_add/', views.add_author, name='authors_add'),  
]
```

Итак, у нас все готово для вывода на экран формы для добавления авторов в БД. Запустим наше приложение и загрузим главную страницу сайта. Пройдем регистрацию для того, чтобы появилась возможность выполнять редактирование сведений об авторах. Теперь по открывшейся ссылке **Редактор авторов** перейдем на страницу редактирования сведений об авторах (рис. 11.57).

**Мир книг - печатные и электронные интерактивные книги**

Главная страница [О компании](#) [Контакты](#)

Список авторов для редактирования

Все авторы	Id	Имя	Фамилия	Добавить автора	
				<a href="#">Изменить</a>	<a href="#">Удалить</a>
Выход	1	Александр	Беляев	<a href="#">Изменить</a>	<a href="#">Удалить</a>
Вошел:	2	Александр	Пушкин	<a href="#">Изменить</a>	<a href="#">Удалить</a>
Тестер_1	3	Лев	Толстой	<a href="#">Изменить</a>	<a href="#">Удалить</a>
Редактор авторов	4	Илья	Ильф	<a href="#">Изменить</a>	<a href="#">Удалить</a>
Мои заказы	5	Евгений	Петров	<a href="#">Изменить</a>	<a href="#">Удалить</a>

Copyright ООО "Интеллектуальные информационные системы". 2023. Все права защищены.

Рис. 11.57. Ссылка для вызова формы добавления нового автора

Если все было сделано правильно, то при переходе по этой ссылке будет загружена форма для добавления в БД нового автора (рис. 11.58).

Как видно из данного рисунка, модули Django, используя теги разметки фреймворка Bootstrap, сгенеририровали HTML-страницу со всеми необходимыми атрибутами:

- метки и поля для ввода текстовой информации («Имя автора», «Фамилия автора»);
- календарь для выбора даты;
- широкое поле для ввода многостраничного текста («Сведения об авторе»);
- кнопку для выбора файла с фотографией (портретом) автора;
- кнопку для сохранения данных.

Заполним поля формы необходимыми данными и нажмем кнопку **Сохранить**. После этого пользователь будет переадресован на страницу со сведениями об авторах, на которой появятся обновленные данные (рис. 11.59).

Теперь реализуем процедуру удаления записи об авторе из БД. Для этого не нужно создавать новую форму, достаточно создать соответствующее представление (view). Откроем файл `views.py` и напишем в нем код для удаления из БД сведений об авторах (листинг 11.33).

Мир книг - печатные и электронные  
интерактивные книги

Главная страница О компании Контакты

Добавить в БД автора книги

Все книги	Имя автора:	<input type="text"/>
Все авторы	Фамилия автора:	<input type="text"/>
Выход	Дата рождения:	<input type="text"/> 18.01.2023 <input type="button" value="..."/>
Вошел: Тестер_1	Сведения об авторе:	<input type="text"/>
Редактор авторов	Фото автора:	<input type="text"/> Выберите файл   Файл не выбран
Мои заказы	<input type="button" value="Сохранить"/>	

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены.

Рис. 11.58. Форма для добавления в БД новых авторов книг

Мир книг - печатные и электронные  
интерактивные книги

Главная страница О компании Контакты

Автор	Фото	Показать фото
Евгений Петров		<input type="button" value="Показать"/>
Борис Зверков		<input type="button" value="Показать"/>
Добавлен автор		
Анатолий Постолит		<input type="button" value="Показать"/>
Добавлен автор		

« Первая Предыдущая Стр. 2 из 2.

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены.

Рис. 11.59. Обновленная страница с добавленными авторами

### Листинг 11.33. Измененный код файла World\_Books\WebBooks\catalog\views.py

```
# удаление авторов из БД
def delete(request, id):
    try:
        author = Author.objects.get(id=id)
        author.delete()
        return HttpResponseRedirect("/edit_authors/")
    except:
        return HttpResponseRedirectNotFound("<h2>Автор не найден</h2>")
```

В эту функцию передается идентификатор записи в БД (`id`). Чтобы перехватить возможную ошибку при удалении данных, используется блок `try...except`. Здесь в переменную `author` заносится идентификатор удаляемой записи и методом `delete()` эта запись удаляется из БД. После этого выполняется возврат на исходную страницу (`edit_authors`).

Создавать ссылку для вызова функции удаления автора не нужно, т. к. мы это сделали ранее в шаблоне `edit_authors.html`. Если вернуться к фрагменту кода этой страницы, то в приведенном далее фрагменте кода строка с соответствующей ссылкой выделена серым фоном:

```
<tbody>
{%
    if author %
        {%
            for author in author.all %}
                <tr>
                    <td>{{author.id}}</td>
                    <td>{{author.first_name}}</td>
                    <td>{{author.last_name}}</td>
                    <td><a href="/edit_author/{{author.id}}">Изменить</a> ,
                        <a href="/delete/{{author.id}}">Удалить</a></td>
                </tr>
{%
    endfor %
}{%
    else %
        <p> В базе данных нет авторов</p>
{%
    endif %
}</tbody>
```

А вот создать маршрут для вызова функции удаления автора из БД необходимо. Для этого открываем модуль `World_Books\WebBooks\catalog\urls.py` и добавляем в него строку из листинга 11.34 (изменения выделены серым фоном).

### Листинг 11.34. Измененный код файла World\_Books\WebBooks\catalog\urls.py

```
from django.urls import path
from .import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(), name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
```

```

path('authors/<int:pk>/', views.AuthorDetailView.as_view(),
      name='authors-detail'),
path('about/', views.about, name='about'),
path('contact/', views.contact, name='contact'),
path('mybooks/', views.LoanedBooksByUserListView.as_view(),
      name='my-borrowed'),
path('edit_authors/', views.edit_authors, name='edit_authors'),
path('authors_add/', views.add_author, name='authors_add'),
path('delete/<int:id>/', views.delete, name='delete'),
]

```

Здесь при вызове функции `delete` в нее в виде целого числа передается идентификатор той записи в БД, которую нужно удалить (`<int:id>`). Теперь при нажатии левой кнопки мыши на ссылке **Удалить** в одной из строк таблицы со сведениями об авторах информация об авторе будет удалена из БД (рис. 11.60).

**Мир книг - печатные и электронные  
интерактивные книги**

Главная страница О компании Контакты

Список авторов для редактирования

Добавить автора

Все книги	Id	Имя	Фамилия	Редактировать
Все авторы	1	Александр	Беляев	<a href="#">Изменить</a> , <a href="#">Удалить</a>
Выход	2	Александр	Пушкин	<a href="#">Изменить</a> , <a href="#">Удалить</a>
Вошел: Тестер_1	3	Лев	Толстой	<a href="#">Изменить</a> , <a href="#">Удалить</a>
Редактор авторов	4	Илья	Ильф	<a href="#">Изменить</a> , <a href="#">Удалить</a>
Мои заказы				

Будет удален  
Лев Толстой

Рис. 11.60. Ссылка на функцию удаления автора из БД

Нам осталось реализовать процедуру корректировки (изменения) сведений об авторах. Для этого мы будем использовать другой базовый класс — `ModelForm()`.

#### 11.6.4. Форма для обновления информации об авторах книг на основе класса `ModelForm()`

В этом разделе мы создадим класс формы для изменения сведений об авторах — `Form_edit_author` на основе базового класса `forms.ModelForm`. Так как эта форма связана с моделью данных, то поля для формы определять не нужно, они будут подключены к форме автоматически, т. е. программный код за счет класса `forms.ModelForm` значительно упрощается.

Открываем файл `views.py`, пишем в нем код для изменения в БД сведений об авторах (листинг 11.35).

#### Листинг 11.35. Измененный код файла `World_Books\WebBooks\catalog\views.py`

```
# добавлено для редактирования авторов
from .models import Author
# форма для изменения сведений об авторах
class Form_edit_author(forms.ModelForm):
    class Meta:
        model = Author
        fields = '__all__'
```

Обратите внимание, насколько прост программный код создания формы на основе класса `ModelForm`. Здесь сначала была импортирована модель данных с информацией об авторах (`from models import Author`). Затем в классе `Form_edit_author` создан подкласс `Meta`, в котором всего две строки:

- `model = Author` — создан объект `model` на основе модели данных `Author`;
- `fields = '__all__'` — подключены все поля, которые имеются в модели данных об авторах.

Теперь необходимо создать представление (`view`), которое будет обрабатывать форму редактирования сведений об авторах. Открываем файл `views.py` и пишем в нем код функции для изменения в БД сведений об авторах (листинг 11.36).

#### Листинг 11.36. Измененный код файла `World_Books\WebBooks\catalog\views.py`

```
# импорт для редактирования автора
from .forms import Form_edit_author
# изменение данных об авторе в БД
def edit_author(request, id):
    author = Author.objects.get(id=id)
    # author = get_object_or_404(Author, pk=id)
    if request.method == "POST":
        instance = Author.objects.get(pk=id)
        form = Form_edit_author(request.POST, request.FILES, instance=instance)
        if form.is_valid():
            form.save()
            return HttpResponseRedirect("/edit_authors/")
    else:
        form = Form_edit_author(instance=author)
        content = {"form": form}
        return render(request, "catalog/edit_author.html", content)
```

Эта функция принимает параметр `id` — идентификатор записи в БД со сведениями об авторе. Здесь сначала создается объект `author`, в который из БД загружаются сведения о конкретном авторе:

```
author = Author.objects.get(id=id)
```

После этого обрабатываются два типа запросов от пользователя: POST (сохранение данных, введенных в форму) и GET (вызов формы для изменения данных).

Если получен GET-запрос, то создается объект form на основе класса Form\_edit\_author, в который загружаются сведения о конкретном авторе:

```
form = Form_edit_author(instance=author)
```

Созданная форма упаковывается в словарь content, который через функцию render передается в шаблон edit\_author.html:

```
return render(request, "catalog/edit_author.html", content)
```

Если же получен POST-запрос, то создается объект instance, в который загружаются данные об авторе с идентификатором в БД (id). Затем создается объект form на основе класса Form\_edit\_author, в который загружаются все данные, введенные пользователем в форме корректировки информации об авторе. Если в данных формы нет ошибок (if form.is\_valid), то обновленные сведения об авторе записываются в БД — form.save(). После этого пользователь возвращается на страницу корректировки сведений об авторах (edit\_authors).

#### ПРИМЕЧАНИЕ

Обратите внимание на строку, в которой создается объект form: form = Form\_edit\_author(request.POST, request.FILES, instance=instance). Так как в форме вводится изображение, то здесь обязательно должен присутствовать параметр request.FILES. При его отсутствии файл с изображением не будет передаваться и не будет записан в папку media/images/.

Нам не хватает шаблона страницы, на которой пользователь сможет редактировать сведения об авторах. Создадим в папке World\_Books\WebBooks\templates\catalog\ файл с именем edit\_author.html и напишем в нем код листинга 11.37.

#### Листинг 11.37. Код файла World\_Books\WebBooks\templates\catalog\edit\_author.html

```
{% extends "catalog/base.html" %}  
{% block content %}  
<div class="row text-start lh-2">  
    <h3>Изменить сведения об авторе</h3>  
</div>  
<form method="post" enctype="multipart/form-data">  
    {% csrf_token %}  
    <table class="text-start">  
        {% for field in form %}  
            <tr>  
                <th>{{ field.label_tag }}  
                    {{ field.errors }}  
                </th>  
                <th>{{ field }}</th>  
            </tr>  
        {% endfor %}  
        <th>  
            <button type="submit">Сохранить</button>  
        </th>
```

```
</table>
</form>
{%- endblock %}
```

Здесь с использованием тегов Bootstrap создана таблица. В первой колонке таблицы выводятся метки полей и сообщения об ошибках, если они возникнут — {{field.label\_tag}} {{field.errors}}. Во второй колонке выводятся сами поля — {{field}}. Для вывода всех полей на основе тега for организован цикл, таким образом, каждое новое поле будет выводиться в новую строку таблицы.

Создавать ссылку для вызова функции изменения сведений об авторе не нужно, т. к. мы это сделали ранее в шаблоне edit\_authors.html. Если вернуться к фрагменту кода этой страницы, то в приведенном далее фрагменте кода строка с соответствующей ссылкой выделена серым фоном:

```
<tbody>
{%- if author %}
  {%- for author in author.all %}
    <tr>
      <td>{{author.id}}</td>
      <td>{{author.first_name}}</td>
      <td>{{author.last_name}}</td>
      <td><a href="/edit_author/{{author.id}}">Изменить</a> ,
          <a href="/delete/{{author.id}}">Удалить</a></td>
    </tr>
  {%- endfor %}
{%- else %}
  <p> В базе данных нет авторов</p>
{%- endif %}
</tbody>
```

А вот создать маршрут для вызова функции изменения сведений об авторе необходимо. Для этого открываем модуль World\_Books\WebBooks\catalog\urls.py и добавляем в него строку из листинга 11.38 (изменения выделены серым фоном).

#### **Листинг 11.38. Измененный код файла World\_Books\WebBooks\catalog\urls.py**

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>', views.AuthorDetailView.as_view(),
         name='authors-detail'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('mybooks/', views.LoanedBooksByUserListView.as_view(), name='my-borrowed'),
    path('edit_authors/', views.edit_authors, name='edit_authors'),
```

```

path('authors_add/', views.add_author, name='authors_add'),
path('delete/<int:id>/', views.delete, name='delete'),
path('edit_author/<int:id>/', views.edit_author, name='edit_author'),
]

```

Здесь при вызове функции `edit_author` в нее в виде целого числа передается идентификатор той записи в БД, которую нужно отредактировать (`<int:id>`).

Итак, у нас все готово для вывода на экран формы для изменения сведений об авторе. Запустим наше приложение и загрузим главную страницу сайта. Пройдем регистрацию для того, чтобы появилась возможность редактировать сведения об авторах. Теперь по открывшейся ссылке **Редактор авторов** перейдем на страницу редактирования сведений об авторах (рис. 11.61).

Список авторов для редактирования				
Добавить автора				
	Id	Имя	Фамилия	Редактировать
Выход Вашел: Тестер_1 Редактор авторов Мои заказы	1	Александр	Беляев	<a href="#">Изменить</a> , <a href="#">Удалить</a>
	2	Александр	Пушкин	<a href="#">Изменить</a> , <a href="#">Удалить</a>
	3	Лев	Толстой	<a href="#">Изменить</a> , <a href="#">Удалить</a>
	4	Илья	Ильф	<a href="#">Изменить</a> , <a href="#">Удалить</a>
	5	Евгений	Петров	<a href="#">Изменить</a> , <a href="#">Удалить</a>

Справка: ООО "Интеллектуальные информационные системы", 2023. Все права защищены.

Рис. 11.61. Ссылка для вызова формы редактирования сведений об авторах

Если все было сделано правильно, то при переходе по ссылке **Изменить** будет загружена форма для редактирования сведений об авторе (рис. 11.62).

Как видно из рис. 11.62, форма не является пустой, она заполнена информацией об авторе, которая загружена из БД. Обратите внимание на поле **Фото автора**, на данной форме сама фотография не отображается, но содержится ссылка на нее, которая имеет синий цвет. Если щелкнуть левой кнопкой мыши по данной ссылке, то откроется окно с полноформатным изображением (рис. 11.63).

В поле `title` на вкладке показан истинный размер изображения, а сама фотография адаптирована под размер окна браузера. При изменении размера окна браузера размер изображения также будет меняться, сохраняя пропорции.

Итак, мы реализовали весь функционал сайта, связанного с добавлением, удалением и корректировкой данных об авторах на основе форм Django. Теперь создадим аналогичные формы для корректировки данных о книгах.

**Мир книг - печатные и электронные  
интерактивные книги**

[Главная страница](#) [О компании](#) [Контакты](#)

Все книги	<b>Изменить сведения об авторе</b>		
Все авторы	<b>Имя автора:</b>	Борис	
Выход	<b>Фамилия автора:</b>	Зверков	
Вошел: Тестер_1	<b>Дата рождения:</b>	16.01.1953	
Редактор авторов	<b>Сведения об авторе:</b>  Кандидат технических наук. доцент. Является автором многочисленных учебных пособий для студентов строительных специальностей, специальных инструкций и методических материалов. Увлекается яхтенным спортом, волейболом, горными лыжами.		
Мои заказы			

**Фото автора:**  На данный момент: **images/Рулевой.jpg**  Очистить  
 Выберите файл | Файл не выбран

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены

Рис. 11.62. Форма для редактирования сведений об авторе

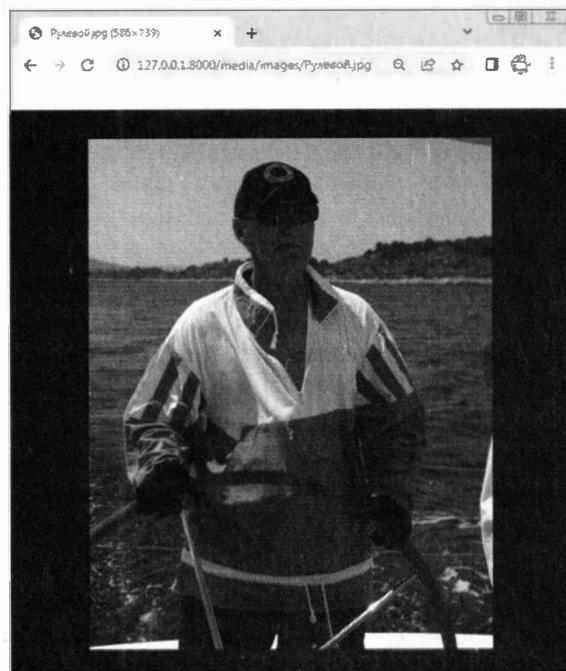


Рис. 11.63. Вывод вкладки с полноформатным изображением автора

## 11.6.5. Форма для ввода и обновления информации о книгах на основе класса *ModelForm()*

Использование базового класса `Form()`, описанное в предыдущем разделе, представляет собой довольно гибкий способ создания форм любой структуры в связке с любой моделью или моделями.

Тем не менее если нужна форма для отображения полей одиночной модели, то это можно сделать проще. Ведь модель уже содержит большую часть информации, которая необходима для построения формы: сами поля, текстовые метки, текст «помощи» и пр. И чтобы не дублировать для создаваемой формы информацию из модели, лучше воспользоваться классом `ModelForm()`, который позволяет создавать формы непосредственно из модели данных. Класс `ModelForm()` можно задействовать в представлениях (`views`) точно так же, как и «классический» класс формы `Form()`.

Итоговая форма при использовании класса `ModelForm()` будет содержать те же поля, что подключенная модель данных. Все, что необходимо сделать внутри создаваемого класса, — это добавить класс `Meta` и связать его с моделью. А затем в поле `fields` указать поля модели данных, которые необходимо включить в форму.

Для начала создадим представление (`view`) и шаблон HTML-страницы, на которую выведем список книг. Это будет таблица, в которой мы создадим ссылки на книги для выполнения следующих действий: добавить новую книгу, удалить книгу, изменить сведения о книге. Здесь нам пока еще не понадобятся формы Django, а сведения о книгах получим из модели.

Для начала создадим представление (`view`), которое будет обрабатывать запросы пользователей. Откроем файл `WebBooks\catalog\views.py` и напишем в нем код листинга 11.39.

### Листинг 11.39. Измененный код файла `World_Books\WebBooks\catalog\views.py`

```
# вызов страницы для редактирования книг
def edit_books(request):
    book = Book.objects.all()
    context = {'book': book}
    return render(request, "catalog/edit_books.html", context)
```

Это очень простое представление. Здесь создается объект `book`, в который из БД загружаются сведения обо всех книгах. Этот объект упаковывается в словарь `context`, который затем передается в шаблон `edit_books.html`.

Создадим теперь в каталоге `World_Books\WebBooks\templates\catalog\` файл с именем `edit_books.html` и напишем в нем код листинга 11.40.

### Листинг 11.40. Код файла `World_Books\WebBooks\templates\catalog\edit_books.html`

```
{% extends "catalog/base.html" %}
{% block content %}
<div class="row text-start lh-2">
    <h5>Список книг для редактирования</h5>
</div>
```

```

<div class="row">
<div class="col-2 "></div>
    <a href="{% url 'book_create' %}">Добавить книгу</a>
<div class="col-8 ">
    <table class="table table-striped table-bordered text-start">
        <thead>
            <tr>
                <th>Id</th>
                <th>Название книги</th>
                <th>Редактировать</th>
            </tr>
        </thead>
        <tbody>
            {% if book %}
                {% for book in book.all %}
                    <tr>
                        <td>{{book.id}}</td>
                        <td>{{book.title}}</td>
                        <td><a href="/book/update/{{book.id}}">Изменить</a> ,
                            <a href="/book/delete/{{book.id}}">Удалить</a></td>
                    </tr>
                {% endfor %}
            {% else %}
                <p> В базе данных нет книг</p>
            {% endif %}
        <tbody>
    </table>
</div>
<div class="col-2"></div>
</div>
{% endblock %}

```

В первой строке выводится заголовок страницы, а во второй строке находится ссылка на страницу с добавлением новой книги:

```
<a href="{% url 'book_create' %}">Добавить книгу</a>
```

Затем с помощью инструкции `if` проверяется условие, есть ли в БД записи с информацией о книгах — `{% if book %}`. Если информация об авторах отсутствует, то в форме будет выдано сообщение «В базе данных нет книг».

Если в БД есть записи о книгах, то далее создается таблица с тремя столбцами:

- идентификатор записи в БД;
- название книги;
- столбец со ссылками на опции редактирования (**Изменить, Удалить**).

Строки таблицы формируются в цикле на основе инструкции `for` — `{% for book in book.all %}`.

Теперь для доступа к данному шаблону нужно прописать маршрут. Открываем модуль `World_Books\WebBooks\catalog\urls.py` и добавляем в него строку из листинга 11.41 (изменения выделены серым фоном).

**Листинг 11.41. Измененный код файла World\_Books\WebBooks\catalog\urls.py**

```
from django.urls import path
from . import views
urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>/', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>/', views.AuthorDetailView.as_view(),
         name='authors-detail'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('mybooks/', views.LoanedBooksByUserListView.as_view(),
         name='my-borrowed'),
    path('edit_authors/', views.edit_authors, name='edit_authors'),
    path('authors_add/', views.add_author, name='authors_add'),
    path('delete/<int:id>/', views.delete, name='delete'),
    path('edit_author/<int:id>/', views.edit_author, name='edit_author'),
    path('edit_books/', views.edit_books, name='edit_books'),
]
]
```

Остался последний шаг, в базовом шаблоне нужно добавить ссылку на страницу edit\_books. Открываем шаблон base.html и добавляем в него строку из листинга 11.42 (изменения выделены серым фоном).

**Листинг 11.42. Измененный код файла \WebBooks\templates\catalog\base.html**

```
{% if user.is_authenticated %}
<a class="nav-link"
 href="{% url 'logout'%}?next={{request.path}}>Выход</a>
<div>Вошел: {{ user.get_username }}</div>
<a class="nav-link"
 href="{% url 'edit_authors' %}>Редактор авторов</a>
<a class="nav-link"
<a class="nav-link"
 href="{% url 'edit_books' %}>Редактор книг</a>
<a class="nav-link"
 href="{% url 'my-borrowed' %}>Мои заказы</a>
{% else %}
<a class="nav-link"
 href="{% url 'login'%}?next={{request.path}}>Вход</a>
{% endif %}
```

Эта ссылка добавлена в боковую панель базового шаблона в блок, к которому имеют доступ только зарегистрированные пользователи. Таким образом, ссылка на страницу с редактированием сведений о книгах откроется только тогда, когда зарегистрированный пользователь войдет в систему.

Протестируем работу сайта после внесенных изменений. После запуска сервера открывается главная страница сайта. Вслед за регистрацией пользователя в системе

в меню боковой панели появится ссылка на страницу для редактирования данных о книгах (рис. 11.64).

Если теперь щелкнуть левой кнопкой мыши по данной ссылке, то откроется страница со списком книг (рис. 11.65).

The screenshot shows the homepage of the 'Mir книг' website. The main title is 'Мир книг - печатные и электронные интерактивные книги'. Below it, a banner says 'На нашем сайте вы можете получить книги в электронном виде'. On the left sidebar, there are links: 'Все книги', 'Все авторы', 'Выход', 'Вошел: Тестер\_1', 'Редактор авторов', 'Редактор книг', and 'Мои заказы'. A tooltip with the text 'Ссылка на страницу редактирования данных о книгах' points to the 'Edit book data' link in the sidebar. The main content area shows statistics: 'Количество книг - 5', 'На складе в наличии - 9', and 'Количество авторов - 7'. Below this, a section titled 'Сведения из базы данных' contains the heading 'Книги' and shows five book covers: 'Двенадцать стульев', 'Золотой теленок', 'Война и мир', 'Дубровский', and 'Продавец воздуха'.

Рис. 11.64. Ссылка на страницу редактирования данных о книгах

The screenshot shows a table titled 'Список книг для редактирования' (List of books for editing) on the 'Mir книг' website. The table has columns: 'Id', 'Название книги' (Book name), and 'Редактировать' (Edit). The data in the table is as follows:

Id	Название книги	Редактировать
1	12 стульев	<a href="#">Изменить</a> , <a href="#">Удалить</a>
2	Золотой теленок	<a href="#">Изменить</a> , <a href="#">Удалить</a>
3	Война и мир	<a href="#">Изменить</a> , <a href="#">Удалить</a>
4	Дубровский	<a href="#">Изменить</a> , <a href="#">Удалить</a>
5	Продавец воздуха	<a href="#">Изменить</a> , <a href="#">Удалить</a>

On the left sidebar, the same navigation links as in Figure 11.64 are present: 'Все книги', 'Все авторы', 'Выход', 'Вошел: Тестер\_1', 'Редактор авторов', 'Редактор книг', and 'Мои заказы'. At the bottom of the page, a copyright notice reads: 'Copyright ООО "Интеллектуальные информационные системы" 2023. Все права защищены'.

Рис. 11.65. Страница со ссылками для редактирования данных о книгах

Как видно из данного рисунка, перед шапкой таблицы находится ссылка **Добавить книгу**, а в каждой строке таблицы две ссылки: **Изменить** — изменить (отредактировать) сведения о книге и **Удалить** — удалить данные о книге из БД. Пока эти ссылки не работают, т. к. не созданы соответствующие страницы. Создадим форму, с помощью которой зарегистрированные пользователи смогут добавлять, редактировать и удалять сведения о книгах. Когда форма создается на основе базового класса `ModelForm()`, то потребуется всего одна форма, как для добавления новой книги, так и для редактирования книги, которая уже занесена в БД. Для удаления книги из БД форма не нужна, поскольку в этом случае новая информация в БД не вводится.

Начнем реализацию формы для работы с книгами с создания класса `BookModelForm()` на основе базового класса `ModelForm()`. Для этого откроем файл `forms.py` и добавим в него код листинга 11.43.

#### Листинг 11.43. Измененный код файла `\WebBooks\catalog\forms.py`

```
# добавлено для редактирования книг
from .models import Book
# форма для изменения сведений о книгах
class BookModelForm(forms.ModelForm):
    class Meta:
        model = Book
        fields = '__all__'
```

Здесь выполнен импорт модели данных `Book`. Затем создан класс `BookModelForm` на основе базового класса `ModelForm()`. В теле этого класса объявлен встроенный класс `Meta`, в котором создан объект `model` на основе модели данных `Book`. Затем создан объект `fields` с указанием тех полей модели, которые будут отображаться на форме. В данном случае на форме будут отображаться все поля из модели данных.

Для каждого поля модели вы можете определить значения меток, виджетов, текстов сообщений об ошибках и т. д., если они не были заданы в модели. Их можно переопределить в классе `meta` при помощи словаря, содержащего поле, которое требуется изменить, и его новые параметры. В качестве примера приведем синтаксис кода, который мы могли бы использовать для изменения параметров поля для ввода аннотации к книге (листинг 11.44).

#### Листинг 11.44. Пример кода изменения параметров полей класса `ModelForm`

```
fields = ['about']
labels = {'about': _('Аннотация'), }
help_texts = {'about': _('Не более 1000 символов'), }
```

Алгоритм управления формами, который мы использовали в функциях представления (`view`) в предыдущем разделе, является примером достаточно общего подхода к работе с формами. Однако Django позволяет значительно упростить программный код за счет применения встроенных базовых классов для создания, редактирования и удаления записей в БД на основе моделей данных.

В этом разделе мы воспользуемся такими классами Django для создания страницы, через которую можно будет добавлять и редактировать данные о книгах. Подобная

страница уже встречалась нам в административной части сайта, когда вводили данные о книгах от имени главного администратора сайта.

Создадим сразу три представления (view) для ввода сведений о новой книге, для обновления сведений о книге и для удаления книги из БД. Откройте файл `WebBooks\catalog\views.py` и добавьте в него код листинга 11.45.

#### Листинг 11.45. Измененный код файла `WebBooks\catalog\views.py`

```
# Импорт для редактирования книг
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy
from .models import Book
# Класс для создания в БД новой записи о книге
class BookCreate(CreateView):
    model = Book
    fields = '__all__'
    success_url = reverse_lazy('edit_books')
# Класс для обновления в БД записи о книге
class BookUpdate(UpdateView):
    model = Book
    fields = '__all__'
    success_url = reverse_lazy('edit_books')
# Класс для удаления из БД записи о книге
class BookDelete(DeleteView):
    model = Book
    success_url = reverse_lazy('edit_books')
```

В первых строках этого модуля выполнен импорт необходимых пакетов.

#### **ПРИМЕЧАНИЕ**

Импорт модели `Book` выделен серым цветом, т. к. мы ранее уже создали строку с импортом этой модели. Здесь эта строка показана только для того, чтобы напомнить про обязательность импорта данной модели.

Здесь на основе базовых классов (`CreateView`, `UpdateView`, `DeleteView`) созданы три пользовательских класса:

- для ввода в БД данных о новой книге — `BookCreate()`;
- для обновления в БД сведений о книге — `BookUpdate()`;
- для удаления книги из БД — `BookDelete()`.

Каждый класс связан с моделью данных (`Book`).

Для классов `BookCreate()` и `BookUpdate()` мы определили показываемые на форме поля (`fields`). Здесь можно было применить тот же синтаксис, что и для класса `ModelForm()` (перечислить все необходимые поля в виде списка), но мы подключили к форме все поля модели данных при помощи инструкции `fields = '__all__'`. При этом если необходимо, то вместо `fields` можно воспользоваться атрибутом `exclude` (от англ. исключить), чтобы определить поля модели, которые не нужно включать в форму. Здесь также можно указать начальные значения для каждого поля, применяя словарь пар «имя\_поля/значение».

По умолчанию наши классы в представлении (`view`) перенаправляют пользователя на страницу, указанную в параметре `success_url`. Это страница, на которую будет перенаправлен пользователь в случае успешного завершения операции, где будут показаны созданные или отредактированные данные. Кстати, при помощи параметра `success_url` можно задать и альтернативное перенаправление.

Классу `BookDelete()` не нужно отображать поля, так что их не требуется и декларировать. Тем не менее необходимо указать параметр `success_url`, потому что для Django не очевидно, что делать после успешного выполнения операции удаления записи. Здесь с параметром `success_url` используется функция `reverse_lazy()` для перехода на страницу списка книг после удаления одной из них. Функция `reverse_lazy()` — это более «кленовая» версия функции `reverse()`. Здесь с помощью функции `reverse_lazy()` указан возврат на страницу редактирования сведений о книгах (`edit_books`).

На следующем шаге мы создадим соответствующие шаблоны. Наши представления (`views`) `BookCreate` и `BookUpdate` по умолчанию будут использовать шаблон с именем `model_name_form.html`. Сuffix в этом имени можно поменять на какой-нибудь другой. Это делается при помощи поля `template_name_suffix` вашего представления (`view`), например, так:

```
template_name_suffix = '_other_suffix'
```

Однако мы не станем так делать, а создадим шаблон HTML-страницы с тем именем, которое в Django задано по умолчанию (в нашем случае это будет шаблон с именем `book_form.html`).

Итак, создайте в каталоге `WebBooks\templates\catalog\` файл с именем `book_form.html` и внесите в него код листинга 11.46.

#### Листинг 11.46. Код файла `WebBooks\templates\catalog\book_form.html`

```
{% extends "catalog/base.html" %}  
{% block content %}  
<form action="" method="post" enctype="multipart/form-data">  
    {% csrf_token %}  
    <table class="text-start">  
        {{ form.as_table }}  
    </table>  
    <input type="submit" value="Сохранить" />  
</form>  
{% endblock %}
```

Код этого шаблона напоминает наши предыдущие формы и прорисовку полей при помощи таблицы. Заметьте, что мы снова указали выражение `{% csrf_token %}`. Кроме того, следует отметить, что код шаблона для класса `ModelForm()` короче и проще, чем код шаблона для класса `Form()`.

#### ПРИМЕЧАНИЕ

В данном шаблоне обратите внимание на тег, в котором создается форма `<form:action="" method="POST" enctype="multipart/form-data">`. Так как в форме вводится изображение, то здесь обязательно должен присутствовать параметр `enctype="multipart/form-data"`. При его отсутствии файл с изображением не будет передаваться и не будет записан в папку `media/images/`.

Представление (`view`) `BookDelete` по умолчанию будет искать шаблон с именем формата `model_name_confirm_delete.html`. Поэтому создайте в каталоге `WebBooks\templates\catalog` файл шаблона с именем `book_confirm_delete.html` и напишите в нем код листинга 11.47.

#### Листинг 11.47. Код файла `WebBooks\templates\catalog\book_confirm_delete.html`

```
{% extends "catalog/base.html" %}
{% block content %}
<h1>Удаление книги</h1>
<p> Вы уверены, что хотите удалить книгу: {{ book }}?</p>
<form action="" method="POST" enctype="multipart/form-data">
    {% csrf_token %}
    <input type="submit" action="" value="Да, удалить" />
</form>
{% endblock %}
```

В заключение откроем файл конфигураций URL-адресов (`WebBooks\catalog\urls.py`) и добавим в него строки листинга 11.48 (добавленные строки выделены серым фоном).

#### Листинг 11.48. Измененный код файла `WebBooks\catalog\urls.py`

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

urlpatterns = [
    path('', views.index, name='index'),
    path('books/', views.BookListView.as_view(), name='books-list'),
    path('books/<int:pk>', views.BookDetailView.as_view(),
         name='book-detail'),
    path('authors/', views.AuthorListView.as_view(), name='authors-list'),
    path('authors/<int:pk>', views.AuthorDetailView.as_view(),
         name='authors-detail'),
    path('about/', views.about, name='about'),
    path('contact/', views.contact, name='contact'),
    path('mybooks/', views.LoanedBooksByUserListView.as_view(),
         name='my-borrowed'),
    path('edit_authors/', views.edit_authors, name='edit_authors'),
    path('authors_add/', views.add_author, name='authors_add'),
    path('delete/<int:id>', views.delete, name='delete'),
    path('edit_author/<int:id>', views.edit_author, name='edit_author'),
    path('edit_books/', views.edit_books, name='edit_books'),
    path('book/create/', views.BookCreate.as_view(), name='book_create'),
    path('book/update/<int:pk>', views.BookUpdate.as_view(),
         name='book_update'),
    path('book/delete/<int:pk>', views.BookDelete.as_view(),
         name='book_delete'),
]
```

Здесь нет ничего нового! Как можно видеть, представления (`view`) являются классами, поэтому они должны вызываться через метод `.as_view()`. Шаблоны URL-адресов для

каждого случая также должны быть вам понятны. Когда поступает запрос book/create (ввести сведения о новой книге), то мы просто обращаемся к представлению views.BookCreate. Когда же поступает запрос на обновление сведений о книге, которая уже занесена в БД (views.BookUpdate), или об удалении конкретной книги (views.BookDelete), то мы обязаны использовать первичный ключ этой книги в БД (pk), чтобы выполнить соответствующую операцию именно с этой книгой.

Итак, у нас все готово для редактирования информации о книгах. Запустим наше приложение и загрузим главную страницу сайта. Пройдем регистрацию для того, чтобы появилась возможность выполнять редактирование сведений о книгах. Теперь по открывшейся ссылке **Редактор книг** перейдем на страницу редактирования сведений о книгах (рис. 11.66).

Все авторы	<b>Id</b>	<b>Название книги</b>	<b>Редактировать</b>
Выход	1	12 стульев	<a href="#">Изменить</a> . <a href="#">Удалить</a>
Вошел: Tester_1	2	Золотой теленок	<a href="#">Изменить</a> . <a href="#">Удалить</a>
Редактор авторов	3	Война и мир	<a href="#">Изменить</a> . <a href="#">Удалить</a>
Редактор книг	4	Дубровский	<a href="#">Изменить</a> . <a href="#">Удалить</a>
Мои заказы	5	Продавец воздуха	<a href="#">Изменить</a> . <a href="#">Удалить</a>

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены

Рис. 11.66. Страница редактирования данных о книгах с активными ссылками

Теперь все ссылки на данной странице являются рабочими. Если щелкнуть левой кнопкой мыши на ссылке **Добавить книгу**, то откроется форма с пустыми полями для ввода сведений о книге (рис. 11.67).

Введем данные о новой книге и нажмем кнопку **Сохранить**. После этого пользователь будет возвращен на страницу с редактированием данных о книгах. На ней уже будет отображена информация о новой книге (рис. 11.68).

Как видно из данного рисунка, после добавления новой книги пользователь возвращается на ту же страницу.

Если теперь щелкнуть левой кнопкой мыши на ссылке **Изменить**, то откроется форма с заполненными полями для редактирования сведений о книге (рис. 11.69).

The screenshot shows a web page titled "Мир книг – печатные и электронные интерактивные книги". The page has a sidebar on the left with links like "Все книги", "Все авторы", "Выход", "Вашел: Тестер\_1", "Редактор авторов", "Редактор книг", and "Мои заказы". The main content area contains several input fields and dropdown menus:

- Название книги:** Text input field with placeholder "Введите название книги".
- Жанр книги:** Dropdown menu with placeholder "Выберите жанр для книги".
- Язык книги:** Dropdown menu with placeholder "Выберите язык книги".
- Издательство:** Dropdown menu with placeholder "Выберите издательство".
- Год издания:** Text input field with placeholder "Введите год издания".
- Автор (авторы) книги:** A dropdown menu containing names: Беляев, Пушкин, Толстой, Ильф. Below it is a placeholder "Выберите автора (авторов) книги".
- Аннотация книги:** A large text area for entering a short description of the book.
- ISBN книги:** Text input field with placeholder "Введите краткое описание книги". Below it is a note "Должно содержать 13 символов".
- Цена (руб.):** Text input field with placeholder "Введите цену книги".
- Изображение обложки:** A file input field with placeholder "Выберите файл" and a note "Файл не выбран". Below it is a placeholder "Введите изображение обложки".
- Сохранить**: A large blue "Save" button at the bottom right.

At the bottom of the page, there is a copyright notice: "Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены."

Рис. 11.67. Страница сайта для ввода в БД информации о новой книге

Как видно из рис. 11.69, это не пустая форма, в нее автоматически загружаются все данные о книге из БД. Обратите внимание на поле **Изображение обложки**. На данной форме сама обложка не отображается, но содержится ссылка на нее, которая имеет синий цвет. Если щелкнуть левой кнопкой мыши по данной ссылке, то откроется окно с полноформатным изображением обложки (рис. 11.70).

В поле `title` на вкладке показан истинный размер изображения, а само изображение адаптировано под размер окна браузера. При изменении размера окна браузера размер изображения также будет меняться, сохраняя пропорции.

Если щелкнуть левой кнопкой мыши рядом с какой-либо из книг на ссылке **Удалить**, то откроется форма, в которой будут присутствовать название выбранной книги и кнопка с надписью **Да, удалить** (рис. 11.71).

**Мир книг - печатные и электронные интерактивные книги**

**Список книг для редактирования**

**Добавить книгу**

<b>Id</b>	<b>Название книги</b>	<b>Редактировать</b>
1	12 стульев	<a href="#">Изменить</a> , <a href="#">Удалить</a>
2	Золотой теленок	<a href="#">Изменить</a> , <a href="#">Удалить</a>
3	Война и мир	<a href="#">Изменить</a> , <a href="#">Удалить</a>
4	Дубровский	<a href="#">Изменить</a> , <a href="#">Удалить</a>
5	Продавец воздуха	<a href="#">Изменить</a> , <a href="#">Удалить</a>
8	Основы программирования на Python (интерактивное цифровое пособие).	<a href="#">Изменить</a> , <a href="#">Удалить</a>

**Сведения о новой книге, добавленной в БД**

Рис. 11.68. Информация о новой книге, добавленной в БД

**Мир книг - печатные и электронные интерактивные книги**

**Главная страница** [О компании](#) [Контакты](#)

**Название книги:**

**Жанр книги:**

**Язык книги:**

**Издательство:**

**Год издания:**

**Автор (авторы) книги:**

**Аннотация книги:**   
Интерактивное цифровое пособие по основам языка программирования Python в первую очередь предназначено для начинающих программистов, которые решили ознакомиться с основами Python с нуля.

**ISBN книги:**   
Должно содержать 13 символов

**Цена (руб.):**   
Введите цену книги

**Изображение обложки:** Изменить:   
Введите изображение обложки

**Сохранить**

Рис. 11.69. Форма для редактирования данных о книге

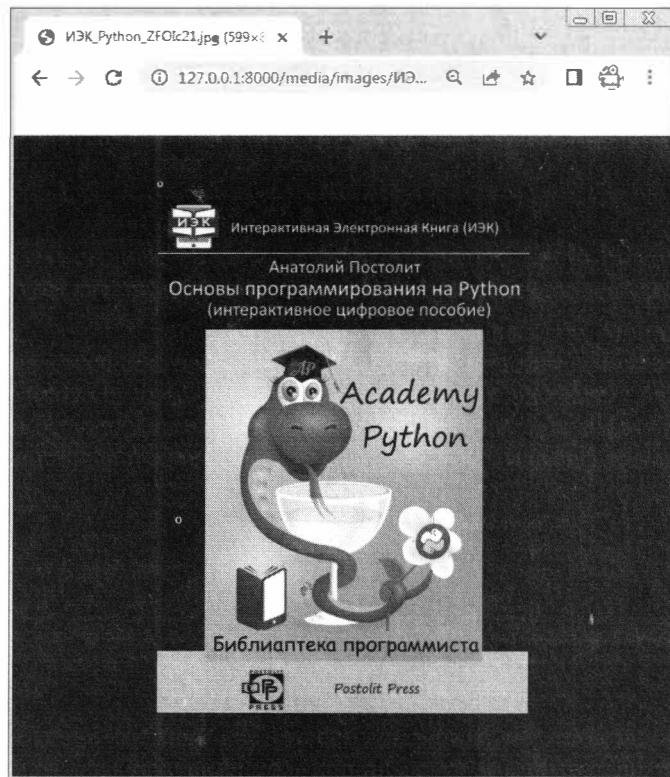


Рис. 11.70. Вывод вкладки с полноформатным изображением обложки книги

Мир книг - печатные и электронные интерактивные книги

Главная страница О компании Контакты

Все книги Удаление книги

Все авторы Вы уверены, что хотите удалить книгу: Основы программирования на Python (интерактивное цифровое пособие).?

Выход Да, удалить

Вашел:  
anatoly

Редактор  
авторов

Редактор  
книг

Мои  
заказы

Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены

A screenshot of a website interface for managing books. On the left, there's a sidebar with links like 'Все книги', 'Все авторы', 'Выход', 'Вашел: anatoly', 'Редактор авторов', 'Редактор книг', and 'Мои заказы'. The main content area has a title 'Удаление книги'. It displays a confirmation message: 'Вы уверены, что хотите удалить книгу: Основы программирования на Python (интерактивное цифровое пособие).?'. There are two buttons at the bottom: a grey 'Да, удалить' button and a white 'cancel' button. At the very bottom of the page, there's a copyright notice: 'Copyright ООО "Интеллектуальные информационные системы", 2023. Все права защищены'.

Рис. 11.71. Страница для удаления из БД информации о книге

Если здесь щелкнуть левой кнопкой мыши кнопку **Да, удалить**, то сведения о книге будут удалены из БД, и пользователь вернется на страницу со списком редактируемых книг.

Итак, мы создали все страницы нашего учебного сайта.

## 11.7. Краткие итоги

Создание и управление формами — это довольно сложная и трудоемкая работа. Однако Django делает этот процесс намного проще за счет встроенных в него готовых механизмов создания, прорисовки и проверки форм. Более того, Django предоставляет встроенные обобщенные классы для создания форм, в которых реализован функционал по созданию, редактированию и удалению записей в БД, связанных с отдельной моделью данных. При этом автоматически генерируются и исполняются SQL-запросы к базе данных.

Сайты в процессе развития и разработки становятся все сложнее. С увеличением числа страниц сайта возрастает количество внутренних взаимодействий между компонентами, в результате чего внесение изменений в одну часть приложения может нарушить работу других его частей. Исходя из этого, разработчику приходится постоянно вручную тестировать и проверять работу всех элементов сайта. Один из способов, который позволяет проверять корректность внесенных изменений, — автоматическое тестирование приложения. В Django реализована возможность автоматизации тестирования создаваемого сайта при помощи специального фреймворка. К сожалению, из-за ограниченного объема книги мы не сможем рассмотреть автоматическое тестирование приложений. Однако вы можете ознакомиться с этим процессом в оригинальной документации на Django.

Итак, мы создали достаточно простой сайт, который позволяет пользователям через веб-интерфейс работать с удаленной базой данных. Теперь необходимо развернуть наш сайт в Интернете. Решению этой задачи посвящена следующая глава.





## ГЛАВА 12

# Публикация сайта в сети Интернет

В предыдущих главах были созданы формы сайта «Мир книг», его общий интерфейс, главная страница, а также страницы для ввода, редактирования и удаления данных из БД со стороны удаленного пользователя.

Теперь, когда мы создали и протестируем свой сайт, необходимо развернуть его на публичном веб-сервере, чтобы он стал доступен через Интернет удаленным пользователям. Эта глава дает общее представление о том, каким образом подойти к поиску хостинга для размещения сайта, что нужно сделать, чтобы разместить свой сайт на публичном сервере.

Мы рассмотрим здесь следующие вопросы:

- как подготовить инфраструктуру сайта перед его публикацией в сети Интернет;
- как выбрать хостинг-провайдера для своего сайта;
- как зарегистрировать аккаунт на сайте хостинг-провайдера;
- как создать виртуальное окружение для размещения сайта на публичном сервере;
- какие изменения нужно внести в параметры настройки веб-приложения;
- как развернуть сайт на хостинге timeweb.

### 12.1. Подготовка инфраструктуры сайта перед публикацией в сети Интернет

Когда разработка сайта завершена, его необходимо разместить на удаленном сервере для публичного доступа. До настоящего момента мы работали в локальном рабочем окружении. При этом использовали веб-сервер Django на собственном компьютере (или в локальной сети) и запускали сайт с небезопасными настройками своей рабочей среды. Перед тем как поместить сайт на публичном сервере, необходимо:

- сделать несколько изменений в настройках проекта;
- выбрать (настроить) окружение для хостинга приложения Django;
- выбрать (настроить) окружение для размещения статических файлов;
- в целях обслуживания сайта настроить инфраструктуру для его развертывания.

В этом разделе сделан обзор мероприятий по выбору хостинга, описан процесс подготовки сайта к публичному размещению, а также приведен практический пример размещения сайта «Мир книг» на хостинге timeweb.

### 12.1.1. Окружение развертывания сайта в сети Интернет

Окружение развертывания — это среда, предоставляемая сервером, на котором вы будете размещать свой веб-сайт для публичного доступа. Такое окружение включает в себя следующие элементы:

- технические средства (серверы), на которых будет работать сайт;
- операционную систему на сервере (Linux, Windows);
- языки программирования времени выполнения (скриптовые) и библиотеки, которые использует ваш сайт;
- веб-сервер для обслуживания страниц и другого контента (Gunicorn, Nginx, Apache);
- сервер приложений, который передает динамические запросы между сайтом Django и веб-сервером;
- базу данных, в которой сайт будет хранить информацию.

Сервер может быть и вашим собственным (с подключением к Интернету по скоростному каналу), но более общим подходом является применение облачных решений. При этом ваш код будет запускаться на удаленном сервере (возможно и виртуальном) на серверной площадке хостинг-провайдера. Хостинг-провайдер обычно предоставляет доступ к своим серверным ресурсам (процессору, оперативной памяти, памяти на жестких носителях, базе данных и т. д.) и соединение с Интернетом за некоторую плату.

Такой тип удаленного доступа к вычислительным средствам называется «Инфраструктура как Сервис» (Infrastructure as a Service, или IaaS). Множество IaaS-поставщиков предлагают услуги по предустановке какой-либо операционной системы, на которую вы сможете установить компоненты, необходимые для вашего рабочего окружения. Другие поставщики предлагают вам выбрать уже готовое полноценное рабочее окружение, возможно включающее в себя Django и настроенный веб-сервер.

Конечно, проще развернуть сайт в уже готовом окружении. Это позволит сделать настройку вашего сайта достаточно простой задачей с минимальным изменением конфигурации. Однако в готовом окружении может быть либо недостаточное количество доступных опций, либо они будут использовать устаревшую операционную систему. Поэтому в некоторых случаях предпочтительнее установить необходимые вам компоненты самостоятельно.

Ряд провайдеров поддерживают Django как часть своего предложения «Платформа как Сервис» (Platform as a Service, или PaaS). При таком хостинге вам не нужно беспокоиться о большей части окружения (веб-сервер, сервер приложений, СУБД и т. п.), т. к. сама платформа берет это на себя (включая все касающееся развития вашего приложения). В подобном случае развертывание приложения является довольно простой задачей — необходимо сконцентрироваться только на своем приложении, а не на внешней инфраструктуре.

Некоторые разработчики выбирают более гибкое решение, предоставляемое IaaS, в то время как другие предпочитают наименьшие накладные расходы и простое масштаби-

рование, предоставляемое PaaS. Когда вы только начинаете работать с веб-приложениями, система типа PaaS представляется более удобной, и именно ее мы будем использовать.

## 12.1.2. Выбор хостинг-провайдера

Существует достаточно много хостинг-провайдеров, которые либо активно поддерживают Django, либо работают с ним. Познакомиться со списком этих компаний можно по следующим ссылкам:

- зарубежные компании — <https://djangofriendly.com/index.html>;
- российские компании — <https://ru.hostings.info/hostings/country/russia>.

Эти поставщики предоставляют различные типы окружений (IaaS, PaaS) и уровни доступа к вычислительным и сетевым ресурсам за разную цену.

Отметим несколько моментов, на которые нужно обратить внимание при выборе хостинга:

- насколько ваш сайт требователен к вычислительным ресурсам;
- уровень поддержки горизонтального масштабирования (добавление большего числа компьютеров) и вертикального масштабирования (переход на более мощные технические средства);
- где находятся серверные площадки и, следовательно, откуда будет получен более быстрый доступ к сетевым ресурсам;
- время непрерывной работы и простоя хостинга;
- инструменты, которые предоставляются для управления сайтом (SSH и FTP), простота и безопасность их применения;
- наличие встроенных фреймворков для работы с вашим приложением;
- наличие ограничений (блокировка электронной почты, число часов «живого» времени за определенную плату, размер места на дисках для данных и т. п.);
- наличие преимуществ (провайдеры могут предложить бесплатные доменные имена и поддержку сертификатов SSL, которые часто необходимо оплачивать);
- что будет с сайтом после окончания времени бесплатного (пробного) использования, какова стоимость платного обслуживания сайта и т. д.?

Существует достаточно много компаний, которые предоставляют пробные бесплатные тарифы типа Evaluation (для пробы), Developer (разработка) или Hobbyist (хобби). Такие сервисы могут быть доступны лишь в течение определенного периода времени. Тем не менее они являются отличным решением для тестирования сайтов с небольшим трафиком в реальном окружении, а также могут при необходимости предоставлять простой доступ к платным ресурсам. Наиболее популярными провайдерами являются Heroku, Python Anywhere, Amazon Web Services, Microsoft Azure и ряд других. Многие провайдеры предлагают базовый тариф (Basic), предоставляющий достаточный уровень вычислительной мощности с небольшим количеством ограничений. Примерами таких провайдеров могут служить Digital Ocean и Python Anywhere. Необходимо помнить, что цена не единственный критерий выбора. Если ваш сайт будет успешен, то

может так случиться, что самым важным элементом при выборе услуг хостинга станет возможность масштабирования.

В настоящее время при выборе зарубежных серверных площадок существует риск быть отключенным от сервиса. В связи с этим предпочтительнее размещать веб-сайты на серверных площадках отечественных компаний. Среди российских компаний можно выделить следующих хостинг-провайдеров: AdminVPS, Timeweb, Sprinthost, Beget.ru, REG.RU и ряд других. В данной книге в качестве примера будет показана публикация сайта на серверах компании Timeweb.

Хостинг timeweb работает более 10 лет, поэтому специалисты компетентны и уверены в своих знаниях. Сервис обеспечивает высокую скорость и максимальную стабильность работы сайтов своих клиентов благодаря современному оборудованию, ПО и программным системам. Для удобства пользователей администрация разработала собственную панель управления. Она хорошо структурирована, а навигация интуитивно понятна. Это отечественная компания с головным офисом в Санкт-Петербурге. Еще одно преимущество использования российского хостинга — поддержка со стороны серверной службы на русском языке. У пользователя есть возможность позвонить в сервисную службу (звонок по России бесплатный из любого региона) или получить ответы на все необходимые вопросы через чат непосредственно на сайте компании. Обычно хостинг-провайдеры предоставляют следующий набор услуг: VPS, выделенные серверы, готовые решения.

- **VPS** (виртуальный частный сервер) — это аренда виртуального сервера с возможностью администрирования самим клиентам. Клиент получает полную свободу в настройке сервера. Ему предоставляется root-доступ, а значит, он может устанавливать любое ПО, редактировать ресурсы, настраивать бэкапы, создавать FTP-аккаунты и др.
- **Выделенные серверы** — это идеальный вариант для высоконагруженных интернет-магазинов, игровых серверов, порталов, у которых высокая посещаемость. Они обеспечивают самую большую производительность и стабильность, а также полную независимость от соседей.
- **Готовые решения** — в данном случае предлагается пропустить настройку сервера и приступить к запуску проекта, используя один из предварительно собранных компанией образов. Можно запустить сервер с одной из таких ОС в один клик: Ubuntu, Fedora, Windows. Также есть возможность запуска сервера со следующими приложениями: MySQL, Django. Для начинающих разработчиков предпочтительнее именно этот вариант хостинга. Следует отметить, что на хостинге timeweb в течение 10 дней можно без оплаты протестировать работу сайта.

## 12.2. Подготовка веб-сайта к публикации

Основа нашего сайта была создана при помощи инструментов Django Admin и manage.py, которые настроены таким образом, чтобы сделать разработку сайта как можно проще. Однако многие настройки, которые хранятся в файле проекта settings.py, перед публикацией сайта должны быть изменены. Это необходимо либо для повышения безопасности, либо для улучшения производительности. Общепринятый подход — создание отдельного файла settings.py, в котором находятся важные параметры настройки. Этот файл будет содержать разные настроочные параметры для разных хостинг-

провайдеров, и они, конечно же, будут отличаться от тех настроек, которые были на этапе разработки сайта.

Критически важны следующие настройки файла `settings.py`:

- `DEBUG` — этот параметр при развертывании сайта должен быть установлен в `False` (`DEBUG = False`). В результате прекратится вывод важной отладочной информации;
- `SECRET_KEY` — это большое случайное число, применяемое для защиты от CSRF (важно, чтобы ключ не указывался в исходном коде и/или не запрашивался с другого сервера). Django рекомендует размещать значение ключа либо в переменной окружения, либо в файле с доступом только для чтения.

Указанную настройку `SECRET_KEY` можно установить с помощью кода листинга 12.1.

#### Листинг 12.1. Пример изменения настроек приложения в файле `settings.py`

```
# Чтение SECRET_KEY из переменной окружения
import os
SECRET_KEY = os.environ['SECRET_KEY']
# ИЛИ чтение ключа из файла
with open('/etc/secret_key.txt') as f:
    SECRET_KEY = f.read().strip()
```

Можно изменить приложение таким образом, чтобы читать `SECRET_KEY` и `DEBUG` из переменных окружения, если те определены, или в противном случае задать для них значения по умолчанию.

Для этого в файле `\WebBooks\settings.py` можно закомментировать значение `SECRET_KEY` и добавить новые строки, выделенные в листинге 12.2 серым фоном и полужирным шрифтом.

#### Листинг 12.2. Пример изменения настроек приложения в файле `settings.py`

```
# SECURITY WARNING: keep the secret key used in production secret!
# SECRET_KEY = 'dhp40_!05cp071e9pd5e5+3_90fev*vq-obx^3^hv8cx0=1#!k'
import os
SECRET_KEY = os.environ.get('DJANGO_SECRET_KEY',
    'cg#p$g+j9tax!#a3cup@1$8obt2_+6k3q+pmu)5%azj6yjkag')
```

В процессе разработки сайта никаких переменных окружения определено не было, таким образом, были применены значения по умолчанию (не имеет значения, какой ключ был у вас в процессе разработки, поскольку при развертывании проекта вам потребуется другой ключ).

Можно закомментировать строку с настройкой `DEBUG`, а под ней добавить новую (листинг 12.3, изменения выделены серым фоном).

#### Листинг 12.3. Пример изменения настроек приложения в файле `settings.py`

```
# SECURITY WARNING: don't run with debug turned on in production!
# DEBUG = True
DEBUG = bool(os.environ.get('DJANGO_DEBUG', True))
```

Значение `DEBUG` будет `True` по умолчанию и станет `False` в том случае, если переменная окружения `DJANGO_DEBUG` будет проинициализирована пустой строкой, т. е. когда `DJANGO_DEBUG = ''`.

#### ПРИМЕЧАНИЕ

Было бы понятнее, если мы могли просто установить для `DJANGO_DEBUG` значение `True` или `False` напрямую, а не использовать «любую строку» или «пустую строку» (соответственно). К сожалению, значения переменных среды хранятся как строки Python, и единственной строкой, которая распознается как `False`, является пустая строка (например: `bool('') == False`).

Мы перед развертыванием сайта не будем менять эти параметры, т. к. они могут нам пригодиться в этом виде при отладке работы сайта после развертывания на сервере. Однако их можно изменить уже после того, как сайт будет успешно запущен на серверной площадке.

## 12.3. Размещение веб-сайта на хостинге timeweb

Для ознакомления с работой проекта в Интернете мы воспользуемся 10-дневным тестовым периодом. Для работы с удаленным сайтом разработчику будет представлена достаточно удобная панель управления. Кроме того, пользователь получает доступ к серверу через SSH-консоль и программные средства, работающие по протоколу FTP. Правда, в течение тестового периода не будет возможности работать с почтой.

### 12.3.1. Регистрация аккаунта пользователя

Итак, приступим. Для начала нужно загрузить главную страницу сайта по следующей ссылке: [timeweb.com/ru](http://timeweb.com/ru). В меню главной страницы нужно выбрать опцию **Виртуальный хостинг** (рис. 12.1).

После этого будет предложено несколько вариантов хостинга (рис. 12.2).

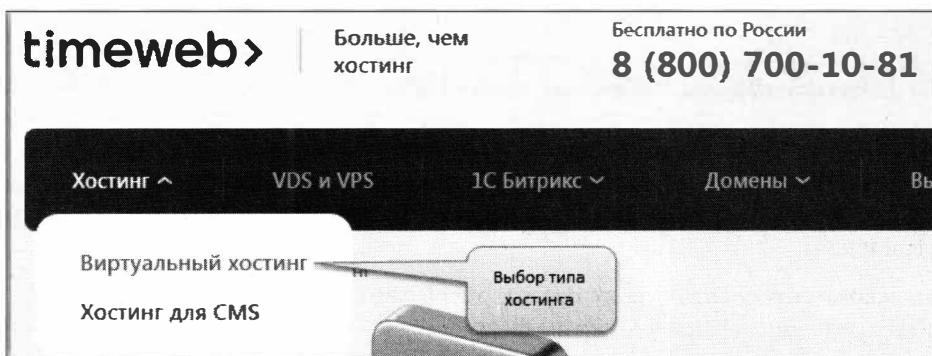


Рис. 12.1. Выбор виртуального хостинга на сайте timeweb

Здесь можно выбрать любой вариант. Однако рекомендуется опция **Optimo+**, которая дает возможность развернуть до 10 сайтов. Это нам пригодится, т. к. в процессе работы нам действительно придется создать как минимум три сайта:

- обычный сайт с параметрами настройки по умолчанию (будет создан автоматически);
- сайт на Django с параметрами по умолчанию (будет нужен для изучения параметров настройки сайтов на Django);
- наш сайт, который будем разворачивать.

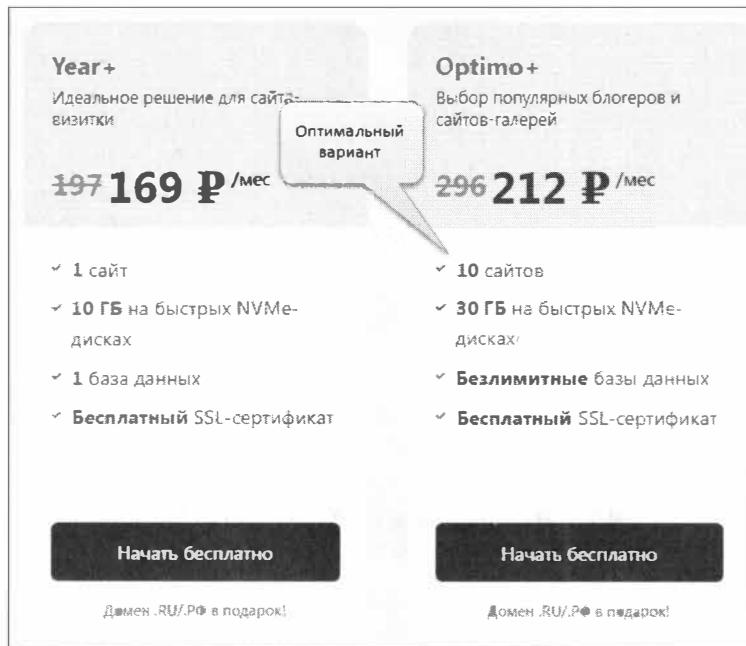


Рис. 12.2. Варианты хостинга

После выбора типа хостинга вам будет предложена страница регистрации аккаунта пользователя (рис. 12.3).

Здесь достаточно ввести ФИО и электронный адрес. Несмотря на подсказку, что ввод номера телефона не обязательен, данное поле нужно заполнить, т. к. это все равно придется сделать после того, как вам откроют доступ на сервер. После нажатия на кнопку **Зарегистрироваться** вам на почту придет уведомление с логином и паролем (рис. 12.4).

После нажатия на кнопку **Войти в панель управления** вы будете сразу направлены на страницу своего личного кабинета. Здесь для начала можно познакомиться с порядком дальнейшей работы. Для этого в верхнем правом углу нужно выбрать опцию **С чего начать работу** (рис. 12.5).

После выбора этой опции откроется окно подсказки (рис. 12.6).

Здесь будет предложено три варианта:

- создать новый сайт;
- перенести сайт с другого хоста;
- разместить файлы сайта с другого компьютера.

## Регистрация

G    VK    O

или заполните форму

Физ. лицо    Юр. лицо

Фамилия, имя и отчество

Email

Телефон

Необязательно

Я хочу указать свое имя пользователя

Код партнера

101515

Зарегистрироваться

Я ознакомлен(а) с политикой, офертой и даю согласие на обработку персональных данных

Рис. 12.3. Страница регистрации



Рис. 12.4. Уведомление об успешной регистрации аккаунта

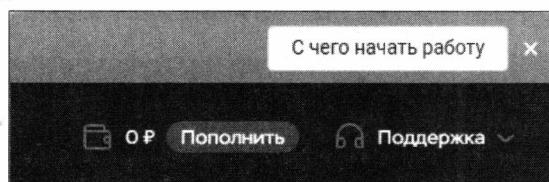


Рис. 12.5. Опция меню С чего начать работу

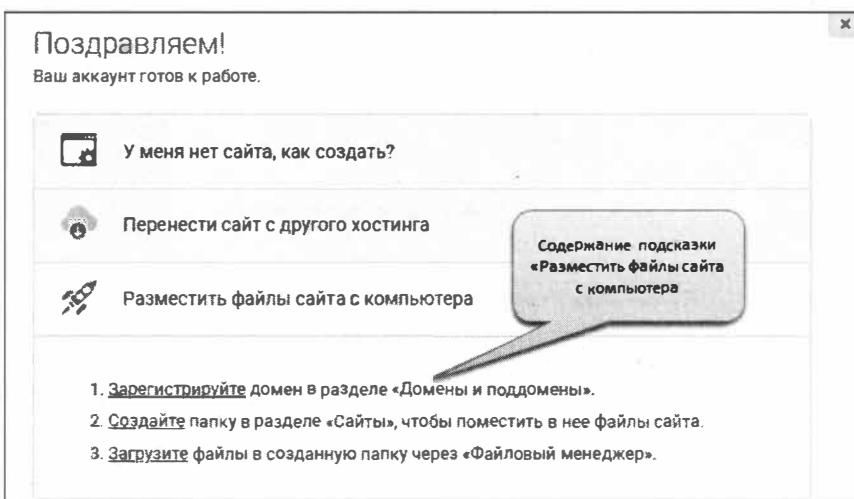


Рис. 12.6. Подсказки о шагах начала работы

После знакомства с этими подсказками можно приступить к работе в административной панели хостинга timeweb.

### 12.3.2. Административная панель хостинга timeweb

У нас сайт уже разработан, поэтому выбираем опцию **Разместить файлы сайта с компьютера**. Регистрировать домен мы пока не будем, воспользуемся тем доменным именем, которое получим на тестовый период 10 дней. Мы создадим новую папку в разделе сайта, инсталлируем в нее необходимые библиотеки и загрузим файлы нашего готового приложения.

После знакомства с подсказками раскрываем вкладку **Сайты** и выбираем опцию **Мои сайты** (рис. 12.7).

Нам будет предоставлен шаблон одностраничного сайта, у которого есть временное доменное имя, действительное в течение 10 дней ([cg95659.tw1.ru](http://cg95659.tw1.ru)). Мы не будем использовать данный шаблон, а создадим новую папку и будем там формировать свой сайт с нуля. Однако воспользуемся данным шаблоном для изучения структуры папок будущего сайта.

Для начала откроем страницу сайта, которая была создана по умолчанию. Для этого щелкнем правой кнопкой мыши на ссылку ([cg95659.tw1.ru](http://cg95659.tw1.ru)) и в открывшемся окне выберем опцию **Открыть ссылку в новой вкладке** (рис. 12.8).

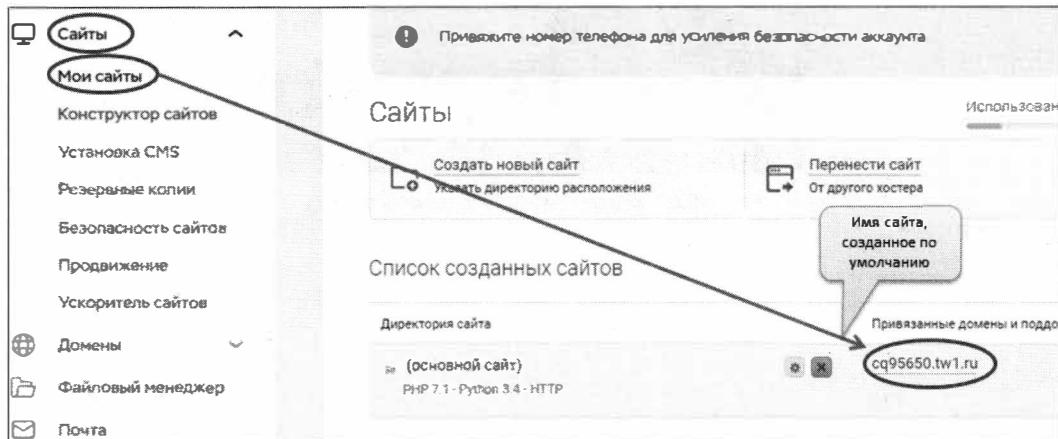


Рис. 12.7. Сайт, созданный по умолчанию

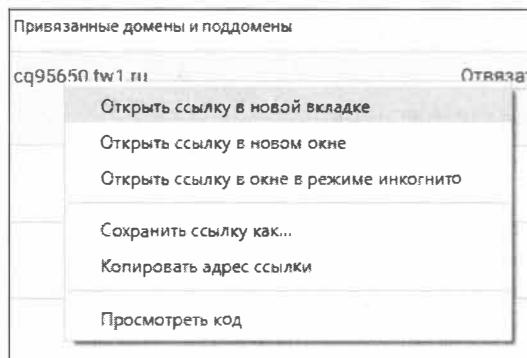


Рис. 12.8. Открытие страницы сайта, созданного по умолчанию

После этого должна открыться главная страница сайта (рис. 12.9).

#### **ПРИМЕЧАНИЕ**

Возможно, перед выдачей этой страницы откроется окно с предложением купить доменное имя. Так как нам в тестовом режиме не нужно покупать доменное имя, то отметьте опцию **Больше не показывать** и нажмите кнопку **Перейти на сайт**.

Теперь посмотрим структуру папок этого сайта и найдем, где находится главная страница. Для этого открываем вкладку **Файловый менеджер**, находим директорию **Основной сайт** и открываем в нем папку с именем `public_html`. В этой папке будет всего один файл с именем `index.html` (рис. 12.10).

Здесь `public_html` — это корневая папка, в которой находятся все файлы сайта. Поскольку это простейший одностраницочный сайт, то в данной папке находится всего один файл с главной страницей — `index.html`.

Убедимся, что это страница именно того сайта, который был создан по умолчанию. Двойным щелчком мыши откроем данную страницу и внесем небольшие изменения в ее код в тег `<body>` сразу после тега `<center>` (рис. 12.11).

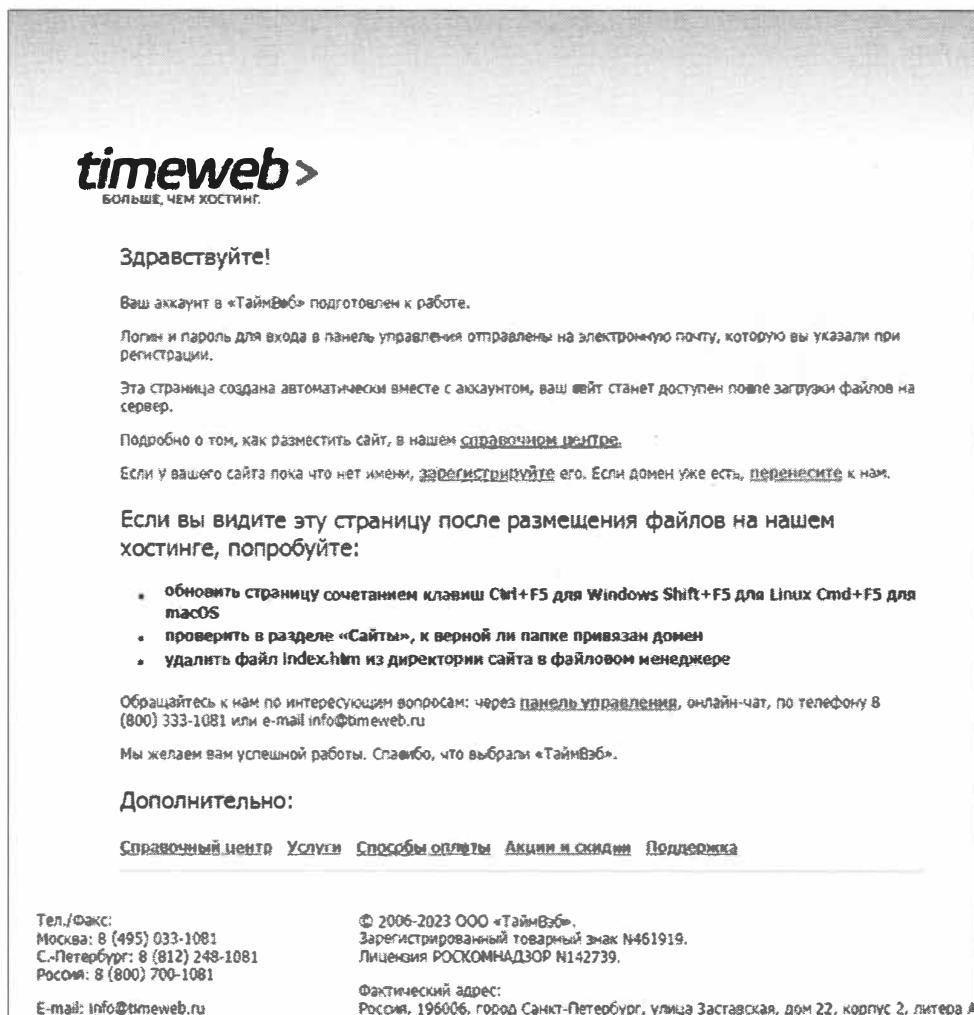


Рис. 12.9. Главная страница сайта, созданного по умолчанию

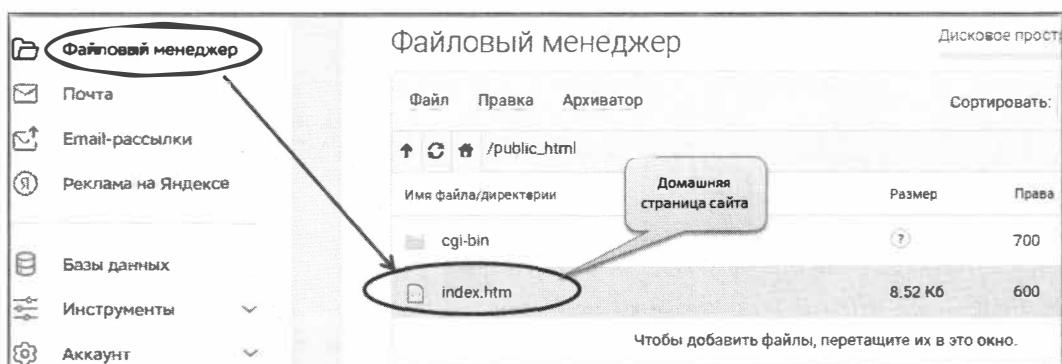


Рис. 12.10. Расположение главной страницы сайта index.html

The screenshot shows a code editor window with the file 'index.htm' open. The code is written in HTML and includes some CSS styles. A callout bubble points to the text 'Этот текст был добавлен на домашнюю страницу сайта' (This text was added to the home page) which is circled in red. Another callout bubble points to the text 'Текст, добавленный на домашнюю страницу сайта' (Text added to the home page). At the bottom left is a 'Сохранить' (Save) button, and at the bottom right are 'Раскрыть на весь экран' (Open full screen) and a help icon.

```

45      }
46
47     p.line {
48         border-top: 1px solid #dbbdbd;
49     }
50   </style>
51 </head>
52
53 <body>
54   <center>
55     Этот текст был добавлен на домашнюю страницу сайта
56     <table border="0" style="width:800px; background-
57         image:url(http://timeweb.com/ru/img/gradient.jpg);background-repeat:repeat-x;background-align:top;">
58       <tr>
59         <td style="height:148px;width:240px;" valign="bottom" align="right"></td>
61         <td>&ampnbsp</td>
62     </tr>
63   </body>

```

Сохранить Для корректного отображения и редактирования файлов в кодировках, отличных от кодировки Unicode, используйте FTP-клиент. Раскрыть на весь экран

Рис. 12.11. Изменения кода домашней страницы index.html

Нажмем кнопку **Сохранить** и закроем данное окно. Если теперь снова откроем наш сайт по ссылке, которая создана по умолчанию, то мы увидим на главной странице сайта внесенные нами изменения (рис. 12.12).

В результате этих действий мы узнали следующее:

- все файлы сайта располагаются в папке public\_html;
- в этой папке находится главная страница сайта index.html.

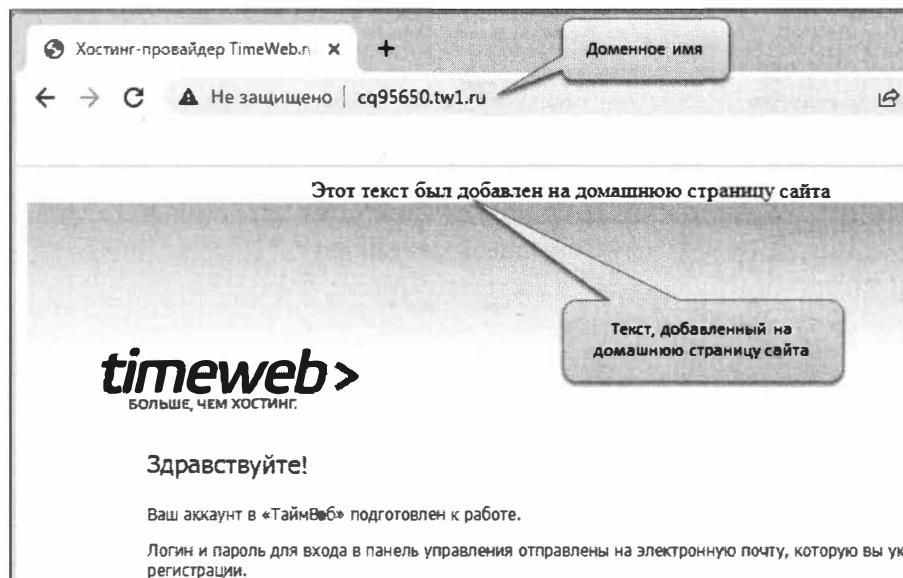


Рис. 12.12. Главная страница сайта с изменениями

Так как у сайтов на Django более сложная структура папок, да еще дополнительные фреймворки и библиотеки, то мы не будем использовать данный шаблон, а создадим пустую директорию и с нуля начнем формировать в ней наш будущий сайт.

### 12.3.3. Создание на сервере папки для нового сайта

Поскольку на серверах приложение будет работать под операционной системой Linux, то нам понадобится терминал (консоль) для того, чтобы мы могли установить на удаленный сервер необходимые нам фреймворки и дополнительные библиотеки. Для пользователей сервиса timeweb в личном кабинете предоставляется и данный терминал (SSH-консоль), и возможность подключения к серверу по протоколу SSH.

#### ПРИМЕЧАНИЕ

SSH (англ. Secure SHell — защищенная оболочка) — сетевой протокол прикладного уровня, предназначенный для безопасного удаленного доступа к UNIX-системам. Преимущество этого протокола в том, что он шифрует всю передаваемую информацию по сети, в отличие от протокола telnet. В основном он нужен для удаленного управления данными пользователя на сервере, запуска служебных команд, работы в консольном режиме с базами данных.

Для создания нового сайта нужно будет выполнить несколько шагов:

- подключиться к серверу;
- создать папку для размещения файлов нашего сайта;
- создать виртуальное окружение для размещения нашего проекта на Django;
- установить в виртуальное окружение Python, Django и другие дополнительные библиотеки.

Практически нам потребуются те же шаги для создания проекта, которые мы делали на локальном компьютере при разработке приложения, только теперь это нужно будет повторить на удаленном сервере. Отличие будет заключаться лишь в том, что на локальном компьютере мы работали в IDE PyCharm и виртуальное окружение было сформировано автоматически при создании нового проекта, а здесь это придется сделать самостоятельно. Поскольку на сервере установлена операционная система Linux, то это нужно будет делать через SSH-консоль с использованием команд Linux. Для тех, кто изначально работал над проектами на Python под Linux, эти команды уже знакомы. Для тех, кто применяет Windows, сложностей не должно возникнуть, т. к. эти команды достаточно просты. После того как мы создадим виртуальное окружение, можно будет переносить разработанное нами приложение на сервер.

Итак, приступим. Для начала нужно активировать SSH-протокол, который необходим для связи с сервером. Для этого в веб-интерфейсе вашего личного кабинета находим панель **Статус сервисов** и активируем переключатель **SSH** (рис. 12.13).

Теперь можно активировать терминальное окно для работы с удаленным сервером. Открываем вкладку **Инструменты** и дважды щелкаем левой кнопкой мыши на опции **SSH-консоль** (рис. 12.14).

После этого у вас откроется окно терминала для работы с удаленным сервером (рис. 12.15).

Консоль откроется в новой вкладке браузера. Подключение произойдет автоматически, вам не потребуется дополнительно вводить какие-либо реквизиты для доступа. Итак,

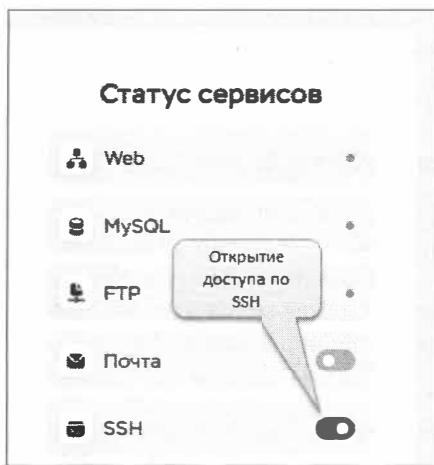


Рис. 12.13. Открытие доступа к серверу по протоколу SSH



Рис. 12.14. Открытие SSH-консоли

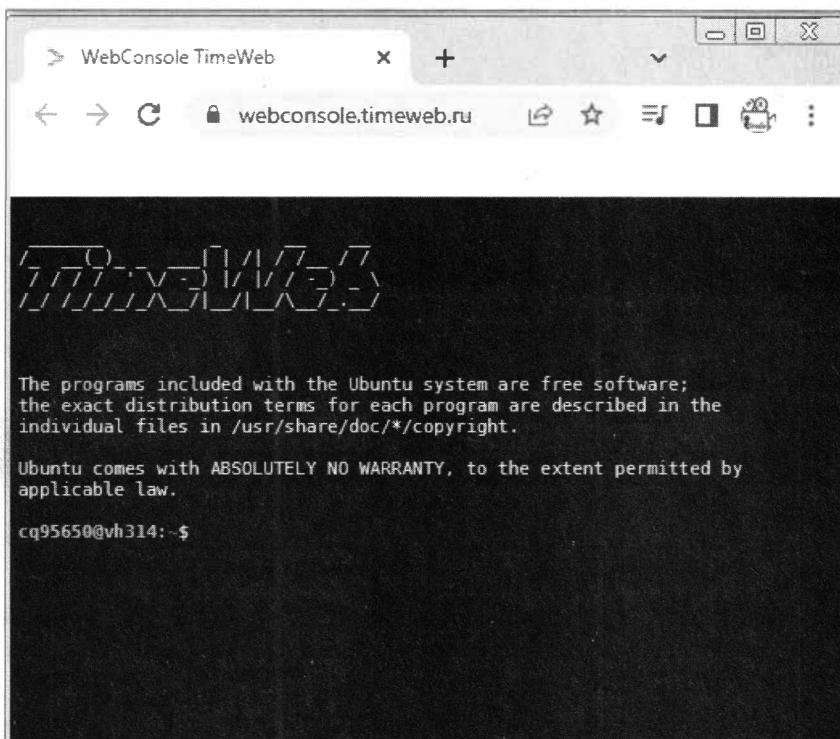


Рис. 12.15. Окно SSH-консоли для работы с удаленным сервером

мы вошли на сервер и теперь можем управлять нашими приложениями. Посмотрим, что же есть в нашей папке. Пишем в окне терминала команду для просмотра содержимого папки:

```
~$ ls
```

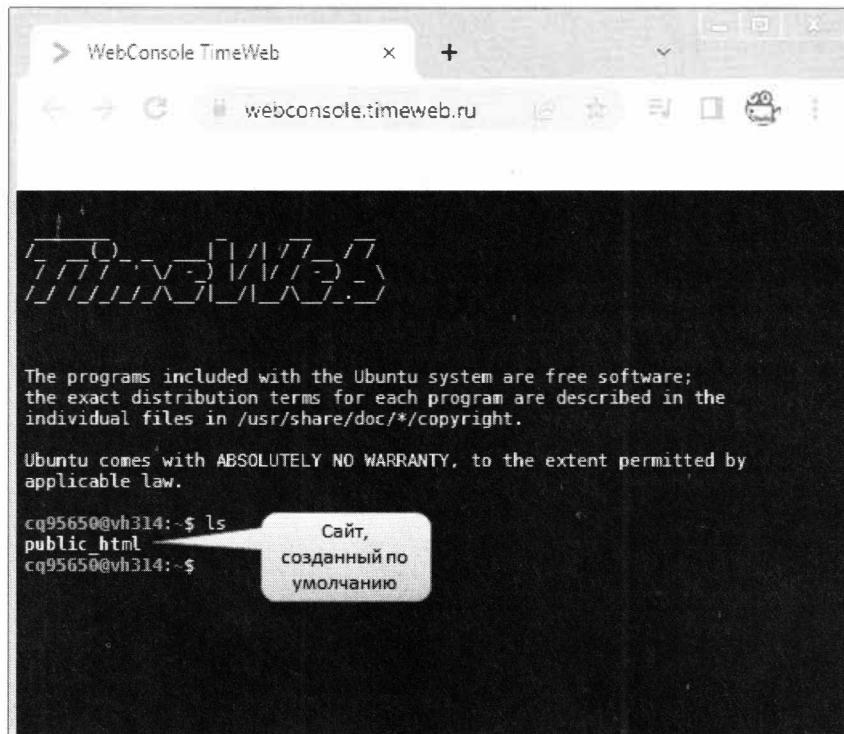


Рис. 12.16. Папка одностраничного сайта, созданного по умолчанию

Мы увидим папку `public_html` нашего одностраничного сайта, который был создан по умолчанию (рис. 12.16).

Итак, мы с использованием веб-страницы нашего личного кабинета посмотрели структуру одностраничного сайта, который был создан по умолчанию, и подключились к корневой папке этого сайта через SSH-консоль. Но данный сайт совсем не то, что нам нужно. Это простой сайт со статичными страницами и без взаимодействия с базами данных.

Мы создадим новый абсолютно пустой сайт, в который будем переносить свое приложение. Для этого открываем вкладку **Сайты** и выбираем опцию **Мои сайты**. В открывшемся окне страницы **Сайты** будет опция **Создать новый сайт** (рис. 12.17).



Рис. 12.17. Запуск процедуры создания нового сайта



Рис. 12.18. Выбор типа создаваемого сайта

После этого появится окно выбора типа создаваемого сайта (рис. 12.18).

Здесь вам будет предложено три варианта:

- создать сайт на основе CMS;
- создать сайт на основе конструктора сайтов;
- создать новую директорию.

Поскольку наш сайт уже разработан и нам его нужно просто перенести на серверную площадку, то мы выбираем опцию **Создать новую директорию**. После этого откроется окно для ввода имени создаваемой директории (рис. 12.19).

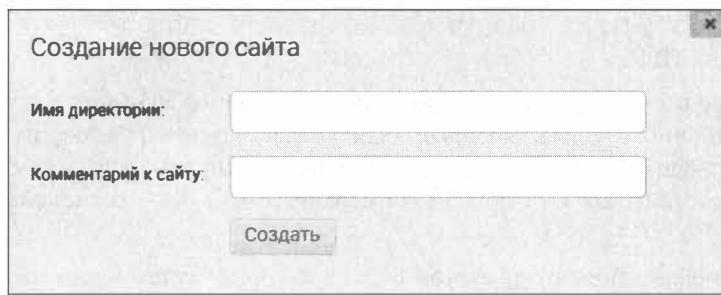


Рис. 12.19. Окно для ввода имени директории создаваемого сайта

Вводим имя нашего сайта «newsite» и нажимаем кнопку **Создать**. После того как завершится процесс формирования пустой папки-шаблона, в списке созданных сайтов появится новый сайт с именем «newsite» (рис. 12.20).

Проверяем, включен ли доступ к удаленному серверу по протоколу SSH (рис. 12.21).

Если вы уже открывали такой доступ, то переключатель должен оставаться во включенном состоянии. Если по каким-либо причинам он выключен, то переведите его во включенное состояние.

Теперь нужно сделать контрольное подключение к нашему новому сайту, созданному на удаленном сервере. Для этого вызываем вкладку **Инструменты** и открываем SSH-консоль (рис. 12.22).



Рис. 12.20. Новый сайт в окне со списком созданных сайтов

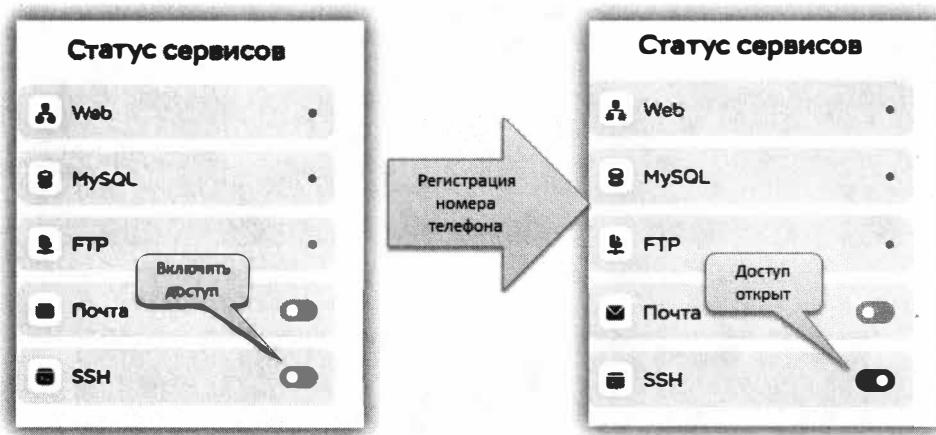


Рис. 12.21. Проверка статуса доступа к удаленному серверу по протоколу SSH

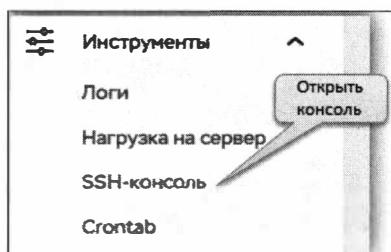


Рис. 12.22. Открытие SSH-консоли

После этого должно появиться терминальное окно, которое подключено к вашему новому сайту на удаленном сервере (рис. 12.23).

Проверяем структуру папок, к которым мы имеем доступ. Пишем в окне терминала команду для просмотра содержимого папки:

```
~$ ls
```

Мы увидим две папки, которые созданы в шаблоне нашего нового сайта: `newsite` и `public_html` (рис. 12.24).



Рис. 12.23. Терминальное окно, подключенное к удаленному серверу

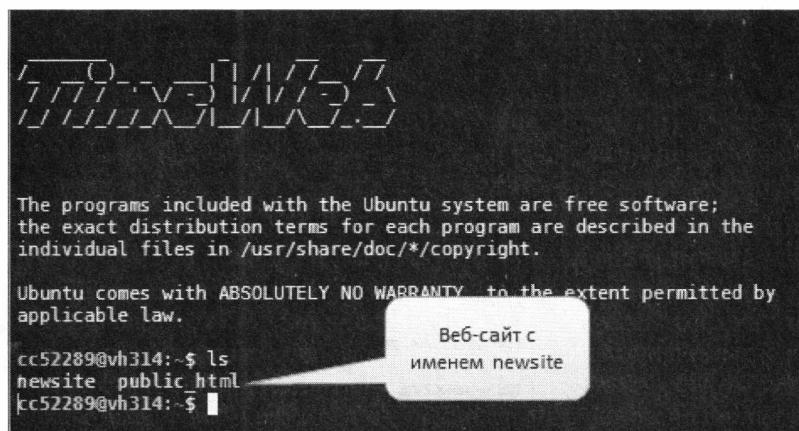


Рис. 12.24. Структура папок сайта с именем «newsite»

Как видно из данного рисунка, пользователь с именем «cc52289» действительно подключен к папке `newsite`, в которую вложена папка `public_html`. Папка `public_html` — это именно та папка, в которую мы будем переносить наше приложение. Если вернуться к нашему проекту, который мы разрабатывали на домашнем компьютере, то это аналог имени проекта, созданного в IDE PyCharm, там наш проект назывался `World_books`.

Теоретически в папке `newsite` уже можно создавать виртуальное окружение и переносить наш проект с домашнего компьютера. Однако мы пока этого делать не будем поскольку мы не знаем, какие параметры и как нужно менять в файле `settings.py` для того, чтобы наш сайт начал корректно работать на новом сервере. Можно пойти долгим путем —

изучать инструкцию данного провайдера по разворачиванию приложений на этом хосте и засыпать вопросами службу поддержки. Но лучше этот путь сократить, подсмотрев новые параметры настройки уже готового сайта Django, развернутого на этой серверной площадке. Мы пойдем именно этим, более коротким путем.

В тестовом режиме можно создать до 10 пробных сайтов, и мы воспользуемся данной возможностью. Создадим еще один сайт с параметрами настройки по умолчанию, но это будет уже шаблон сайта на Django. Для этого воспользуемся шаблоном сайта на Django из каталога CMS. Для новичков нужно раскрыть сущность термина CMS (не путать с термином sms — отправка коротких сообщений по сотовой связи).

CMS (*англ. Content Management System*) или «Система управления контентом» (еще ее иногда называют «движком» сайта) — это набор программного обеспечения для создания и управления сайтом. Естественно, использование CMS значительно ускоряет процесс разворачивания полноценного сайта и облегчает жизнь тем, кто занимается этим впервые. Разобраться с настройками, конечно же, придется, но это займет намного меньше времени. Работа в среде готового набора инструментария не означает, что ваши возможности будут ограничены. CMS поддерживают расширения, так что на такой платформе можно развернуть все, от небольшого блога до солидного интернет-магазина.

Мы воспользуемся системой управления контентом, которая адаптирована для приложений на Django. При этом мы не будем разворачивать наше приложение в шаблоне сайта на Django, созданном по умолчанию с помощью сервиса CMS. Этот шаблон нам нужен будет для того, чтобы посмотреть значения настроек параметров, специфичных именно для выбранного хостинга. Кроме того, мы воспользуемся временным доменным именем этого шаблона, которое будет активным в течение десяти дней тестового периода.

Итак, приступим. В своем личном кабинете timeweb открываем вкладку **Сайты** и жмем левой кнопкой мыши на опцию **Установка CMS**. В списке предлагаемых CMS открываем вкладку **Прочие** и находим там систему управления контентом Django (рис. 12.25).

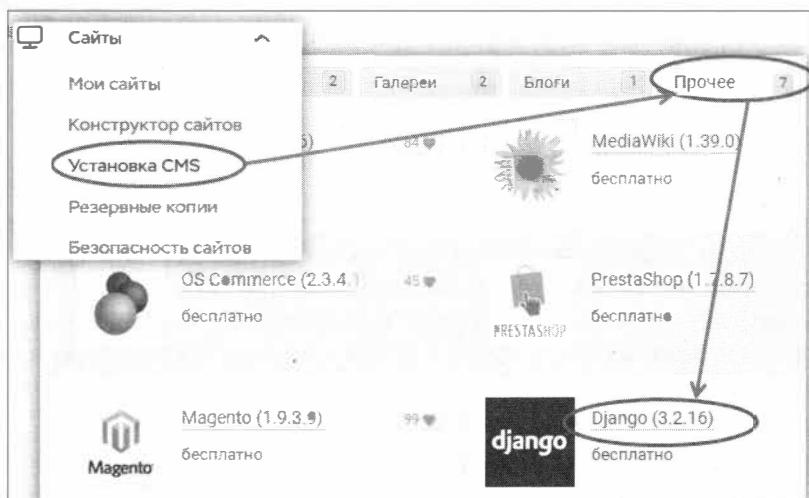
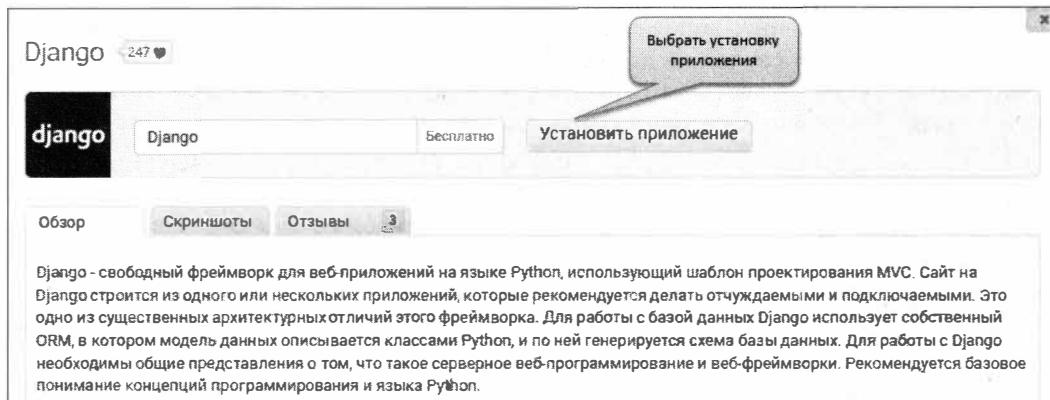


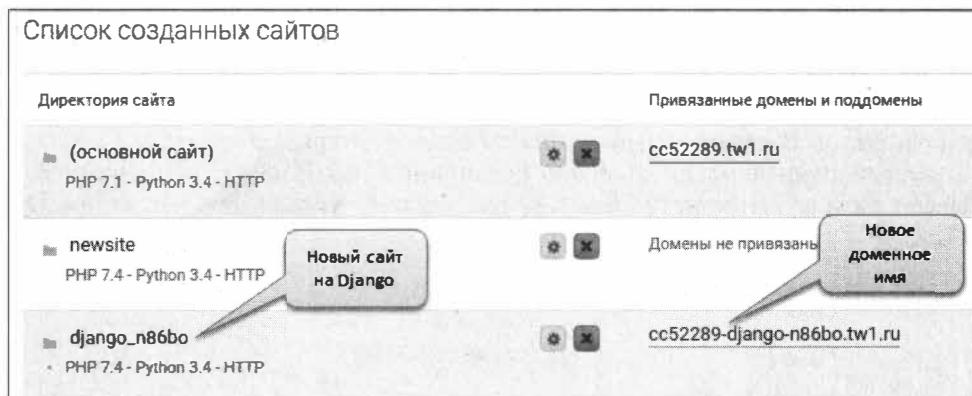
Рис. 12.25. Система управления контентом Django



**Рис. 12.26. Установка системы управления контентом Django**

Если теперь щелкнуть левой кнопкой мыши по ссылке **Django**, то откроется окно с предложением установить данное приложение (рис. 12.26).

Нажимаем на кнопку **Установить приложение** и ждем завершения данного процесса. После этого открываем вкладку **Сайты** и выбираем опцию **Мои сайты**. В списке созданных сайтов появится новый сайт, в имени которого будет ключевое слово «Django» (рис. 12.27).



**Рис. 12.27. Шаблон сайта на Django, созданный по умолчанию**

Как видно из рис. 12.27, был создан новый сайт и для него сгенерировано имя «`django_n86bo`». Кроме того, создано временное доменное имя «`cc52289-django-n86bo.tw1.ru`». Это доменное имя будет существовать в течение десяти дней. Обратите внимание на привязку доменных имен: у сайта на Django есть временное доменное имя, а у нашего сайта «`newsite`» нет временного доменного имени. Отвязем временное доменное имя от сайта «`django_n86bo`» и привяжем его к нашему новому сайту «`newsite`». Для смены привязки доменных имен воспользуемся соответствующими ссылками (рис. 12.28).

Сначала отвязываем доменное имя от сайта «`django_n86bo`», для этого активируем ссылку **Отвязать домен**. Потом привязываем освободившееся доменное имя к нашему сайту через ссылку **Привязать домен** (рис. 12.29).

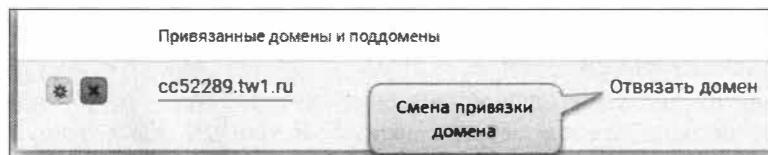


Рис. 12.28. Ссылки для смены привязки доменных имен

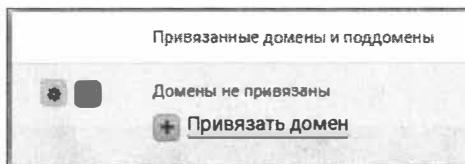


Рис. 12.29. Привязка доменного имени

После этих действий наш новый сайт с именем «newsite» получит временное доменное имя (рис. 12.30).

Как видно из рис. 12.30, временное доменное имя теперь привязано к нашему сайту «newsite», и его можно будет использовать для проверки работы сайта в Сети. Пока мы сделали первый шаг, в результате которого:

- на удаленном сервере создана папка `newsite`, в которой мы развернем наш сайт;
- к нашему сайту привязано временное доменное имя, которое в течение 10 дней позволяет отлаживать сайт.

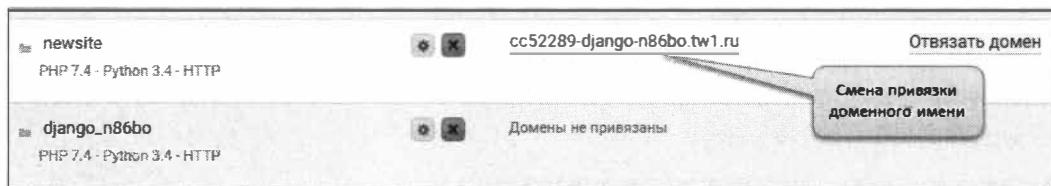


Рис. 12.30. Результаты смены привязки доменного имени

Теперь можно приступить созданию виртуального окружения и созданию в нем приложения Django.

#### 12.3.4. Создание на сервере виртуального окружения и приложения Django для нового сайта

На следующем этапе нужно сделать три шага:

- создать виртуальное окружение;
- установить в это виртуальное окружение необходимое программное обеспечение;
- создать проект Django.

Начнем с создания виртуального окружения `virtualenv` — инструмента, позволяющего создавать изолированные среды для отдельных проектов Python, решая тем самым проблему зависимостей и совместимости приложений разных версий. После установки `virtualenv` вам становится доступен пакет `pip` для установки библиотек Python.

В зависимости от ваших задач установку можно выполнить в домашнюю директорию или в директорию конкретного сайта.

Так как на удаленном сервере наше приложение будет работать под управлением Linux, то нам нужно подключиться к серверу через SSH-консоль. Если она у вас закрыта, нужно снова ее открыть. После того как откроется консоль, нужно войти в директорию `newsite`, для чего в окне терминала выполнить команду

```
~$ cd newsite
```

После этого в терминальном окне будет отображен вход в данную директорию (рис. 12.31).



Рис. 12.31. Вход в директорию `newsite` через SSH-консоль

Далее выполняем следующие шаги.

Скачиваем виртуальное окружение, указав в команде нужную версию Python:

```
~$ wget https://bootstrap.pypa.io/virtualenv/3.6/virtualenv.pyz
```

Создаем виртуальное окружение для нашего проекта, выполнив следующую команду (здесь `venv` — имя папки для виртуального окружения):

```
~$ python3 virtualenv.pyz venv
```

Активируем виртуальное окружение командой

```
~$ source venv/bin/activate
```

Устанавливаем фреймворк Django:

```
~$ pip install django
```

Так как мы в своем проекте использовали еще две библиотеки (`pillow` и `django_cleanup`), то их тоже нужно установить:

```
~$ pip install pillow
~$ pip install django_cleanup
```

Переходим в папку `public_html`:

```
~$ cd public_html
```

В этой папке создаем проект Django с произвольным именем. Когда мы вели разработку веб-приложения, наш проект назывался `WebBooks`. Так как именно его мы будем переносить, то на удаленном сервере создадим проект Django с таким же именем. Выполним следующую команду:

```
~$ django-admin.py startproject WebBooks
```

Посмотрим, действительно ли проект WebBooks был создан. Для этого в окне терминала выполним команду просмотра структуры папок:

```
~$ ls
```

После этого в окне терминала мы должны получить сообщение о том, что проект успешно создан (рис. 12.32).

```
svenv) cc52289@vh314:~/newsite/public_html$ django-admin.py startproject WebBook
(senv) cc52289@vh314:~/newsite/public_html$ ls
cgi-bin  index.htm  WebBooks  ————— Создан проект.
(senv) cc52289@vh314:~/newsite/public_html$
```

Рис. 12.32. Проект с именем WebBooks в папках сайта newsite/public\_html

На данном этапе мы добились следующих результатов:

- создали виртуальное окружение;
- загрузили в виртуальное окружение необходимый софт: Python, Django, pillow, django\_cleanup;
- создали проект Django с именем WebBooks.

Теперь можно закрыть SSH-консоль и вернуться в веб-панель управления личного кабинета timeweb. Здесь можно посмотреть структуру папок нашего сайта с помощью файлового менеджера (рис. 12.33).

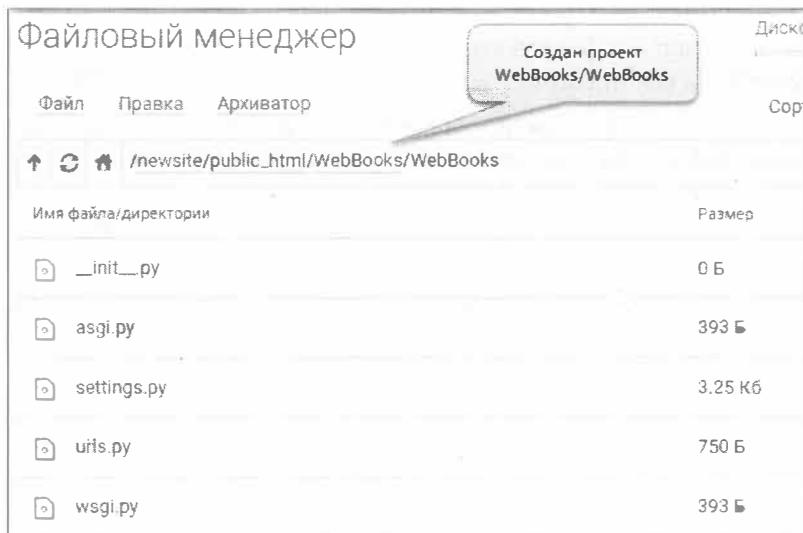


Рис. 12.33. Проект Django с именем WebBooks, созданный на удаленном сервере

Файловый менеджер, который находится в личном кабинете сайта, — это достаточно удобный инструмент для редактирования содержимого файлов. Здесь можно создать, удалить файл или отредактировать его содержимое. Если посмотреть на структуру папок проекта, то она полностью повторяет структуру проекта, который был создан на вашем локальном рабочем компьютере.

В файловом менеджере есть возможность работы с архивами файлов и папок (рис. 12.34).

Архиватор сайта позволяет:

- перенести на удаленный сервер архив проекта с вашего рабочего компьютера;
- распаковать архив на удаленном сервере;
- создать архив вашего сайта, развернутого на сервере;
- скачать архив действующего сайта с удаленного сервера на рабочий компьютер.

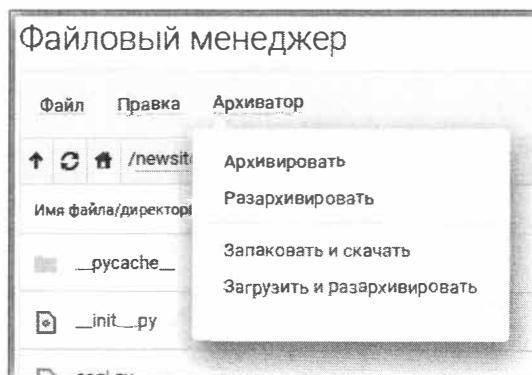


Рис. 12.34. Архиватор файлового менеджера

С использованием данного файлового менеджера создадим новый файл и изменим параметры настройки в некоторых файлах.

Для начала через данный файловый менеджер переходим в папку `newsite/public_html`, создаем там новый файл с именем `.htaccess`. Сразу создать файл с таким именем не получится, т. к. к любому созданному файлу автоматически добавляется расширение. Поэтому сначала создаем файл с именем `.htaccess.html`, а потом переименовываем его, задав новое имя `.htaccess`. Обратите внимание, что имя файла начинается с точки.

Что должно содержаться в данном файле, мы можем подсмотреть в сайте Django, который был создан на основе шаблона CMS (в нашем случае это сайт с именем «`django_n86bo`»). Переходим на данный сайт, заходим в папку `public_html`, открываем файл с именем `.htaccess`. Он будет содержать следующий код (рис. 12.35).

Копируем код этого файла в буфер и затем вставляем его в аналогичный файл нашего проекта. Теперь в этом коде делаем небольшие изменения (листинг 12.4, изменения выделены серым фоном).

#### Листинг 12.4. Код файла `.htaccess`

```
Options +ExecCGI
AddDefaultCharset utf-8 AddHandler wsgi-script.py
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ WebBooks/WebBooks/wsgi.py/$1 [QSA,L]
```

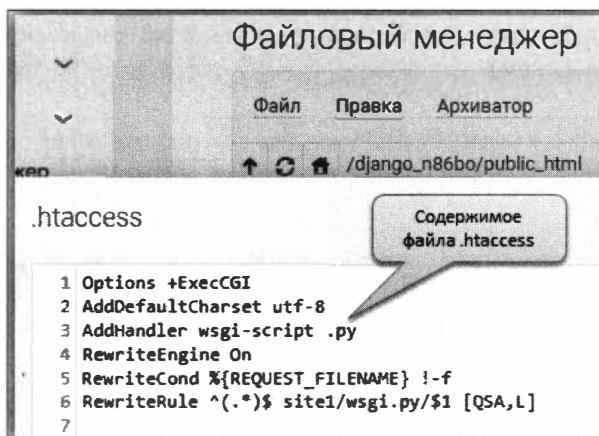
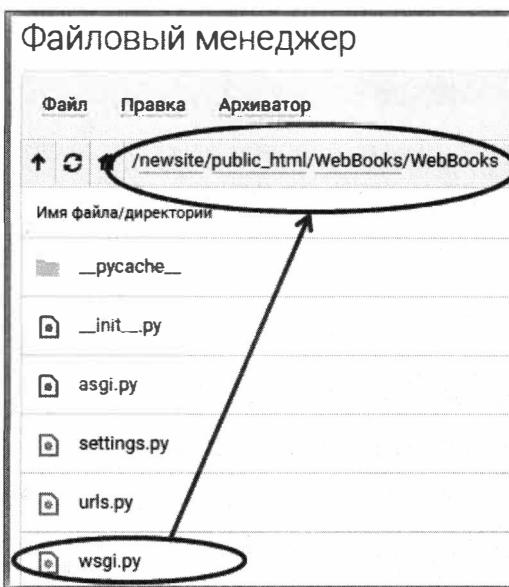


Рис. 12.35. Код файла .htaccess сайта Django

Здесь мы фактически указали путь к файлу `wsgi.py`. Именно по этому пути он находится в папках нашего проекта `WebBooks`. Нажимаем кнопку **Сохранить** и закрываем данное окно. В файловом менеджере можно проверить, что файл `wsgi.py` находится именно там, где требуется (рис. 12.36).

Рис. 12.36. Место расположения файла `wsgi.py`

Затем в этой же папке находим файл `settings.py`, открываем его и вносим следующие минимальные изменения. В самом начале этого модуля добавляем импорт дополнительного пакета (листинг 12.5, изменения выделены серым фоном).

#### Листинг 12.5. Измененный код файла `settings.py`

```

import os
from pathlib import Path

```

Находим строку `ALLOWED_HOSTS = []` и меняем ее, как показано в листинге 12.6 (изменения выделены серым фоном).

#### Листинг 12.6. Измененный код файла `settings.py`

```
ALLOWED_HOSTS = ["*"]
```

Здесь необходимо указать используемое доменное имя (или список допустимых доменных имен). На период отладки мы вместо имени поставили звездочку, означающую, что данный сайт допускает использование любых доменных имен. Когда реальное доменное имя будет получено и зарегистрировано, этот параметр можно будет поменять.

Нам осталось отредактировать содержимое файла `wsgi.py`, который находится в этой же папке. Что должно содержаться в данном файле, мы снова можем подсмотреть на сайте Django, который был создан на основе шаблона CMS (в нашем случае это сайт с именем «`django_n86bo`»). Переходим на данный сайт, заходим в папку `public_html/site1`, открываем файл с именем `wsgi.py`. Он будет содержать следующий код (рис. 12.37).

```

Файловый менеджер
Файл Правка Архиватор
/django_n86bo/public_html/site1
wsgi.py

1 # -*- coding: utf-8 -*-
2
3 import os
4 import sys
5 import platform
6
7 #путь к проекту
8 sys.path.insert(0, '/home/c/cc52289/django_n86bo/public_html')
9 #путь к фреймворку
10 sys.path.insert(0, '/home/c/cc52289/django_n86bo/public_html/site1')
11 #путь к виртуальному окружению
12 sys.path.insert(0, '/home/c/cc52289/django_n86bo/django/lib/python{0}/site-
    packages'.format(platform.python_version()[0:3]))
13 os.environ["DJANGO_SETTINGS_MODULE"] = "site1.settings"
14
15 from django.core.wsgi import get_wsgi_application
16 application = get_wsgi_application()
17

```

Рис. 12.37. Код файла `wsgi.py` сайта Django

В этом файле нужно будет изменить четыре строки. Копируем код этого файла в буфер. Открываем аналогичный файл `wsgi.py` на нашем сайте, удаляем содержимое этого файла и вставляем код из буфера. Теперь корректируем этот файл, как показано в листинге 12.7 (изменения выделены серым фоном).

**Листинг 12.7. Измененный код файла wsgi.py**

```
# -*- coding: utf-8 -*-
import os
import sys
import platform
# путь в проекте к модулю manage.py
sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks')
# путь в проекте к модулю settings.py
sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks/WebBooks')
# путь к виртуальному окружению
sys.path.insert(0, '/home/c/cc52289/newsite/venv/lib/python{0}/site-
packages'.format(platform.python_version()[0:3]))
os.environ["DJANGO_SETTINGS_MODULE"] = "WebBooks.settings"
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

В данном файле мы указали реальные пути на удаленном сервере к следующим файлам:

- к модулю `manage.py`;
- к модулю `settings.py`;
- к виртуальному окружению (`venv`);
- к параметрам настройки.

Теперь в нашем проекте данный файл будет иметь следующий вид (рис. 12.38).

Остался один шаг — проверить работу установленного приложения Django. Если нажать левой кнопкой мыши на временное доменное имя (в нашем случае это `cc52289-`

wsgi.py

```
1 # -*- coding: utf-8 -*-
2
3 import os
4 import sys
5 import platform
6
7 #путь к проекту там, где manage.py
8 sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks')
9 #путь к фреймворку там где settings.py
10 sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks/WebBooks')
11 #путь к виртуальному окружению
12 sys.path.insert(0, '/home/c/cc52289/newsite/venv/lib/python{0}/site-
packages'.format(platform.python_version()[0:3]))
13 os.environ["DJANGO_SETTINGS_MODULE"] = "WebBooks.settings"
14
15 from django.core.wsgi import get_wsgi_application
16 application = get_wsgi_application()
17
```

**Рис. 12.38. Код файла wsgi.py нового сайта с именем newsite**

`django-n86bo.tw1.ru`) или открыть эту ссылку в новой вкладке браузера, то должна загрузиться административная панель Django (рис. 12.39).

Если вы увидели подобную картину, то это значит, что все действия на предыдущих этапах выполнены правильно. Теперь можно перейти к следующему этапу — переносу вашего сайта с рабочего компьютера на удаленный сервер.

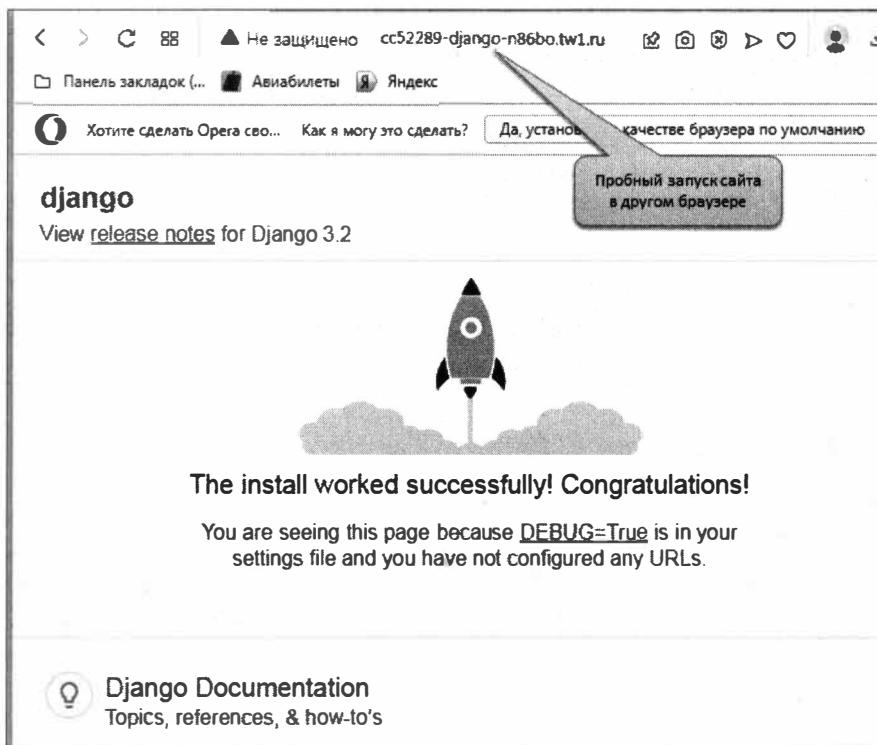


Рис. 12.39. Административная панель Django при активации временного доменного имени

### 12.3.5. Перенос сайта с рабочего компьютера на удаленный сервер

Итак, теперь можно приступить к переносу веб-приложения с локального рабочего компьютера на удаленный сервер. Для переноса данных нам понадобится файловый менеджер, который позволяет взаимодействовать с удаленными компьютерами по протоколу FTP. Мы будем применять достаточно распространенный файловый менеджер Total Commander, который позволяет выполнять практически все операции с файлами и папками: создавать, удалять, копировать, редактировать. Этот файловый менеджер осуществляет обмен данными через протокол FTP.

#### ПРИМЕЧАНИЕ

FTP (англ. File Transfer Protocol) — протокол передачи файлов по сети. Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером. Пользователи FTP могут пройти аутентификацию, передавая логин и пароль открытым текстом, или же, если это разрешено на сервере,

ре, они могут подключиться анонимно. Еще один протокол — SSH — служит для безопасной передачи, шифрующей (скрывающей) логин и пароль, а также содержимое файла.

Параметры подключения к серверу по FTP-протоколу можно найти в панели управления на вкладке **Дашборд** в разделе **Доступ по FTP** (рис. 12.40).

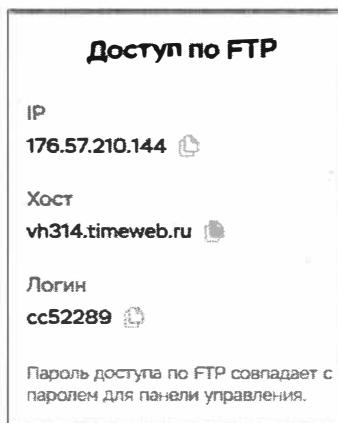


Рис. 12.40. Параметры FTP доступа к удаленному серверу

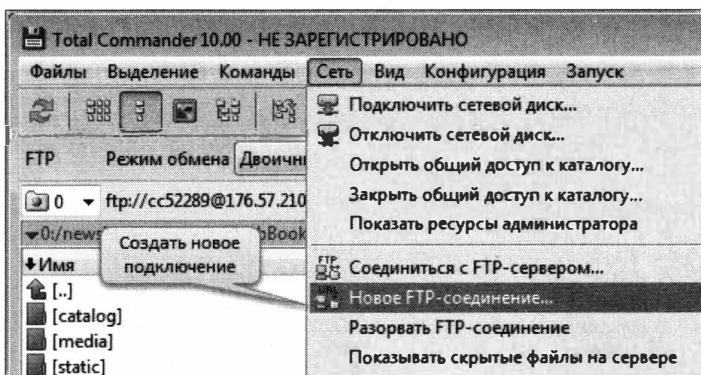


Рис. 12.41. Создание нового соединения в окне файлового менеджера

Открываем файловый менеджер и в меню **Сеть** выбираем опцию **Новое FTP-соединение** (рис. 12.41).

В открывшемся окне вводим параметры удаленного сервера и нажимаем кнопку **OK** (рис. 12.42).

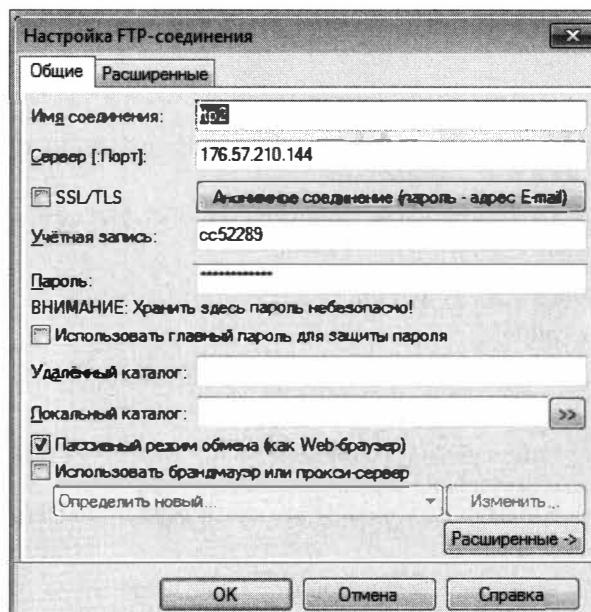


Рис. 12.42. Открытие подключения к удаленному серверу

После этого откроется окно, в котором будут две панели (рис. 12.43).

Как видно из рис. 12.43, в левой панели открыты папки с нашим проектом WebBooks на удаленном сервере, а в правой панели открываем папку с проектом WebBooks на локальном компьютере, где велась разработка проекта.

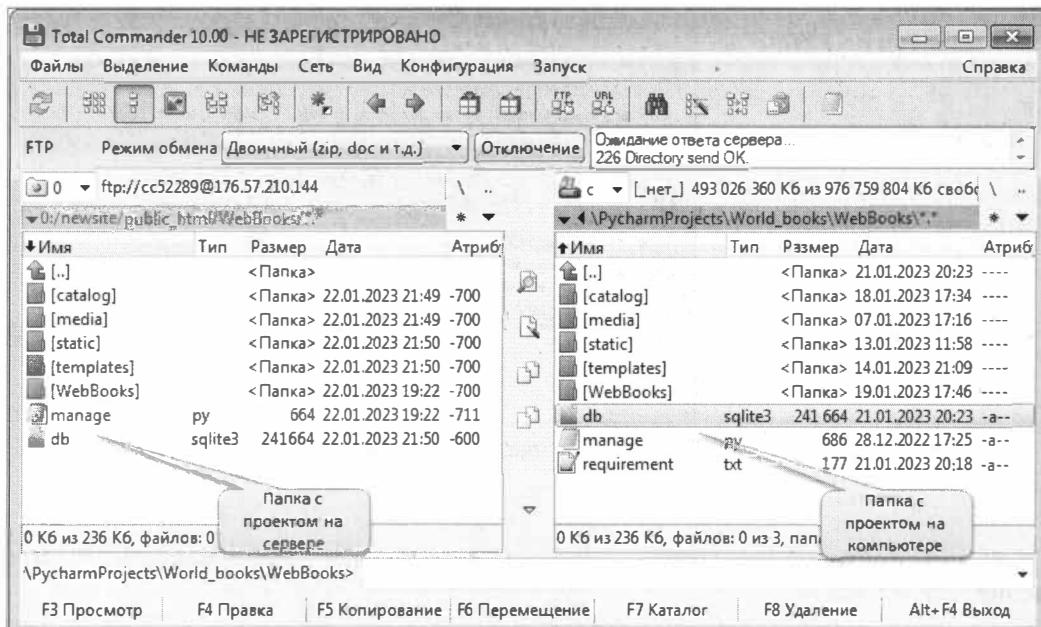


Рис. 12.43. Окно файлового менеджера Total Commander

Теперь с использованием файлового менеджера копируем на удаленный сервер следующие папки:

- catalog — там находится наше приложение;
- media — в этой папке находятся медиафайлы, которые пользователи могут загружать удаленно (изображения и документы);
- static — файлы с изображениями, скрипты и CSS-файлы (в том числе файлы Bootstrap);
- templates — шаблоны HTML-страниц;
- db — база данных sqlite3.

Все эти папки и файлы копируем и переносим с рабочего компьютера на удаленный сервер.

Теперь в левой и в правой панелях открываем папку WebBooks/WebBooks. В нашем проекте в данной папке находится файл urls.py. Переносим этот файл с локального компьютера на удаленный сервер. На этом процесс переноса файлов нашего сайта с локального компьютера на сервер завершен.

Теперь нужно снова вернуться к файловому менеджеру личного кабинета на сайте timeweb и перенести папку static в другую директорию. На нашем локальном компьютере эта папка находилась по следующему пути: WebBooks/static. На удаленном сервере она

должна находиться в папке: newsite/public\_html/static/. Копируем папку static из директории: newsite/public\_html/WebBooks/static и вставляем ее в директорию newsite/public\_html/static/. Это изменение продемонстрировано на рис. 12.44.

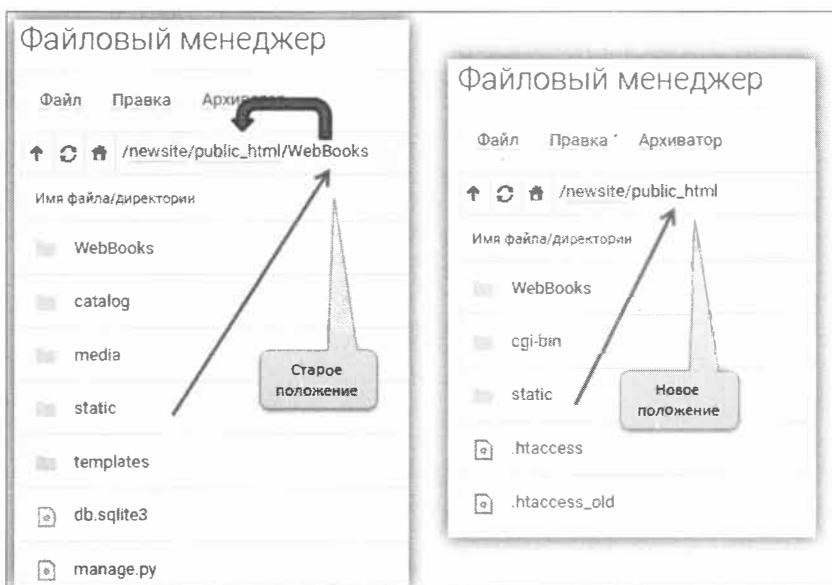


Рис. 12.44. Изменение расположения папки static

Как видно из данного рисунка, папку static нужно поднять на один уровень вверх (все вложенные папки с их содержимым тоже должны переместиться).

Осталось внести некоторые изменения в файл WebBooks/WebBooks/settings.py. Открываем данный файл на удаленном сервере и меняем строки листинга 12.8 (изменения выделены серым фоном).

#### Листинг 12.8. Измененный код файла settings.py

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

Здесь мы в параметр INSTALLED\_APPS добавили наше приложение (catalog) и пакет, предназначенный для очистки хранилища медиафайлов (django\_cleanup).

Далее добавляем строки, указывающие место расположения шаблонов HTML-страниц (листинг 12.9, изменения выделены серым фоном).

### Листинг 12.9. Измененный код файла settings.py

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR, ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
],
]
```

Указываем новый путь к папке static и путь к медиафайлам (листинг 12.10, изменения выделены серым фоном).

### Листинг 12.10. Измененный код файла settings.py

```
STATIC_URL = '/static/'
STATIC_ROOT = '/home/c/cc52289/newsite/public_html/static'
# STATICFILES_DIRS = [
#     # os.path.join(BASE_DIR / "static"),
# ]
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

В конце файла добавляем строки листинга 12.11 (изменения выделены серым фоном).

### Листинг 12.11. Измененный код файла settings.py

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
# Переадресация на главную страницу сайта после входа в систему
LOGIN_REDIRECT_URL = '/'
# настройки отправки e-mail
''' это пробная отправка на консоль '''
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
''' это реальная отправка через smtp@gmail '''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
# EMAIL_USE_TLS = True
EMAIL_USE_SSL = True # для отправки с gmail
EMAIL_PORT = 465
EMAIL_HOST_USER = "postolit1952@gmail.com" # от кого
EMAIL_HOST_PASSWORD = "yvgdwpjklzjmzhlo" # пароль почты отправителя
```

Здесь указан параметр для переадресации на главную страницу (`LOGIN_REDIRECT_URL = '/'`) и параметры отправки электронной почты внешним пользователям для смены пароля.

После всех этих изменений наш сайт должен быть доступен в Интернете по временному доменному имени (в нашем случае это `cc52289-django-n86bo.tw1.ru`). Для чистоты эксперимента копируем эту ссылку и вставляем ее в адресную строку интернет-браузера. Если никаких ошибок не было, то должна открыться главная страница нашего сайта (рис. 12.45).



Рис. 12.45. Главная страница сайта, загруженная с удаленного сервера

Если ваша страница загрузилась, значит, можно оплачивать услуги хостинга, зарегистрировать доменное имя и заменить временное доменное имя на постоянное.

## 12.3.6. Смена временного доменного имени на постоянное

Итак, мы развернули свой сайт на удаленном сервере. Теперь оплачиваем услуги хостинга. Лучше оплачивать размещение сайта сразу на год, в этом случае экономия составит порядка 28%. Оплатить услуги хостинга можно через личный кабинет, там же можно заказать и доменное имя и оплатить его регистрацию. Заказ доменного имени можно сделать на вкладке **Домены**, опция **Купить домен** (рис. 12.46).



Рис. 12.46. Регистрация доменного имени

Здесь вы можете подобрать свободное доменное имя и оплатить его регистрацию. После того как вам будет выделено доменное имя, на электронную почту придет соответствующее письмо (рис. 12.47).

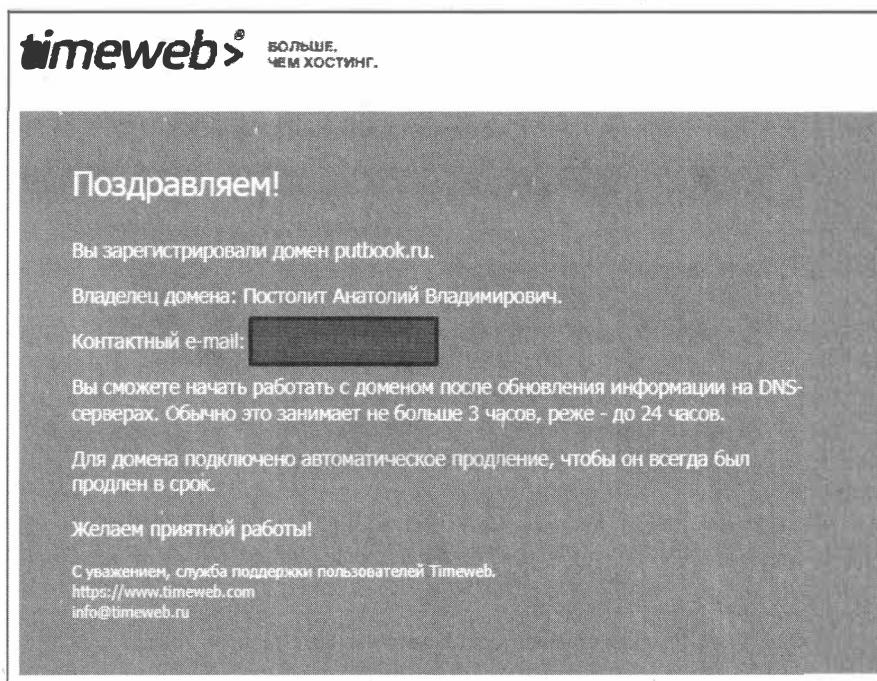


Рис. 12.47. Письмо с подтверждением регистрации доменного имени

В данном случае зарегистрировано доменное имя **putbook.ru**. Теперь можно в личном кабинете привязать ваш сайт к зарегистрированному доменному имени. Если теперь в панели управления на сайте хостинг-провайдера открыть вкладку **Мои сайты**, то там появится ссылка на зарегистрированное доменное имя (рис. 12.48).

По умолчанию оно будет привязано к одному из созданных сайтов. Если оно окажется привязанным не к нашему сайту с именем «*newsite*», то нужно будет отвязать домен, после чего привязать к нашему сайту.

После привязки зарегистрированного доменного имени сайт не сразу появится в Интернете. Нужно будет подождать некоторое время (при создании данного примера сайт

заработал через 4 часа). По истечении некоторого времени сайт станет доступен пользователям сети Интернет.

Итак, мы выполнили все шаги от проектирования сайта до развертывания его в Интернете. Теперь учебный сайт, разработанный в данной книге, можно будет посмотреть по адресу [putbook.ru](http://putbook.ru).

Директория сайта	Привязанные домены и поддомены	
❑ (основной сайт) PHP 7.1 - Python 3.4 - HTTP	cc52289.tw1.ru	Отвязать домен
❑ newsite PHP 7.4 - Python 3.4 - HTTP	cc52289-django-n86bo.tw1.ru putbook.ru	Отвязать домен
❑ django_n86bo PHP 7.4 - Python 3.4 - HTTP	Домены не привязаны	

Рис. 12.48. Зарегистрированное доменное имя в списке созданных сайтов

## 12.4. Краткие итоги

В этой главе были рассмотрены вопросы публикации веб-сайта в сети Интернет. Эта процедура весьма сложная, но необходимая. Сложность ее заключается в том, что приходится вносить много изменений в настройки уже готового веб-приложения. Кроме того, требуется создать на удаленном сервере необходимое окружение, подгрузить в это окружение нужные библиотеки и дополнительные модули, перенести базу данных. Хотя Django и берет на себя автоматизацию некоторых из этих процессов, все равно разработчик должен ряд настроек выполнить вручную, что требует определенных знаний и квалификации. Особенно много трудностей возникает у новичков при первой попытке публикации сайта. Поэтому рекомендуется получить навыки публикации сайта на достаточно простом приложении с минимальным числом страниц и простейшей базой данных.

Мы разрабатывали сайт с использованием базы данных SQLite, и эту же базу данных, состоящую из единственного файла, перенесли в Интернет. Она будет успешно работать в том случае, если к ней будет подключено небольшое число пользователей. Но для того, чтобы обеспечить приемлемое быстродействие при наличии многих пользователей сайта, базу данных нужно переносить на одну из СУБД: PostgreSQL или MySQL. Как это можно сделать, будет показано в следующей главе.



## ГЛАВА 13



# Приложения Django и MySQL

В предыдущих главах был создан сайт «Мир книг» с использованием СУБД SQLite, его общий интерфейс, главная страница, а также страницы для ввода, редактирования и удаления данных из БД со стороны удаленного пользователя.

Теперь, когда мы поработали с «легкой» СУБД SQLite, нужно научиться создавать сайты на основе более мощных СУБД, таких как MySQL или PostgreSQL, т. к. именно эти СУБД чаще всего применяются на публичных серверах. В этой главе подробно описаны все шаги, от разработки сайта на локальном компьютере до его развертывания на публичном сервере с использованием MySQL. Мы рассмотрим здесь следующие вопросы:

- как выбрать СУБД перед публикацией сайта в Интернете;
- как установить сервер MySQL на локальный компьютер;
- как создать базу данных на MySQL на локальном компьютере и подключить к ней проект Django;
- как сформировать инфраструктуру на удаленном сервере для развертывания сайта Django с СУБД MySQL;
- как создать базу данных MySQL на удаленном сервере;
- как перенести сайт с локального компьютера на публичный сервер.

### 13.1. Подготовка инфраструктуры сайта для перехода на MySQL

При работе с базами данных у нас есть много доступных СУБД. Мы можем выбирать между реляционными базами данных, такими как MySQL, PostgreSQL, SQL Server, SQLite, MariaDB, и нереляционными базами данных или базами данных, отличными от SQL, такими как MongoDB и Redis Couchbase. Поскольку Django — это полноценный фреймворк, он совместим практически со всеми базами данных. Официальный Django поддерживает PostgreSQL, MariaDB, MySQL, Oracle и SQLite. Для подключения других СУБД, возможно, придется проделать дополнительную работу или задействовать некоторые плагины.

В предыдущей главе мы развернули наш сайт на хостинге timeweb, который поддерживает работу сайтов с СУБД MySQL, поэтому именно на эту базу данных мы будем переносить наш сайт. Рекомендуется изначально в процессе разработки проекта выбрать именно ту СУБД, которая будет использоваться на хостинге. Мы работали с SQLite, т. к. для новичков при изучении Django это оптимальный вариант. Теперь, когда уже есть некоторый опыт создания веб-приложений, перенесем наш сайт на СУБД MySQL. Для того чтобы не испортить результаты предыдущих трудов, мы начнем разработку приложения с нуля. Но сначала нужно установить сервер MySQL на локальный компьютер.

## 13.2. Инсталляция сервера MySQL

Для начала нам нужно установить сервер MySQL на свой локальный компьютер. Процедура установки для ОС Windows и Linux будет отличаться. Поскольку разработка проекта велась на Windows, мы выполним загрузку СУБД в эту операционную систему. В Интернете кроме официального сайта, где обычно выкладывается самая последняя версия, можно найти множество ссылок для скачивания дистрибутива. В народе не зря родилась пословица: «Лучшее враг хорошего». В подтверждение этому самая свежая версия MySQL, скачанная с официального сайта, начала выдавать ошибки. Поэтому здесь приводится ссылка на сайт, на котором находился дистрибутив одной из предыдущих версий, работающий без ошибок:

<https://kmsauto2020.ru/Allprograms/mysql-8-0-20/>.

В процессе установки вам будет показано с десяток окон. Мы их здесь приводить не будем, т. к. это типичные действия при инсталляции приложений. Тем более что в Интернете можно найти множество соответствующих инструкций. Вот ссылка на одну из них:

<https://info-comp.ru/sisadminst/448-installing-mysql-5-6-23-windows-7.html>.

Итак, на вашем рабочем компьютере установлен сервер MySQL, и можно приступить к созданию базы данных.

## 13.3. Создание базы данных

Прежде чем приступить к настройке подключения проекта Django к БД, убедитесь, что в вашей системе запущен сервер MySQL, есть учетная запись и создана база данных, к которой вы хотите подключиться. Если базы данных еще нет, то ее нужно создать. Это можно сделать несколькими способами:

- из программы, написанной на Python;
- с использованием терминала MySQL Shell;
- с помощью программной оболочки MySQL Workbench.

С появлением в составе MySQL программы MySQL Workbench процесс создания баз данных значительно упростился. То, что раньше приходилось делать вручную с помощью SQL-скрипта из командной строки терминала, теперь можно сделать «в визуальном режиме» с помощью дружественного графического интерфейса.

Создать новую БД можно двумя способами: через панель **SCHEMAS** или через панель инструментов главного меню.

Для того чтобы создать новую БД через панель **SCHEMAS**, нужно в области этой панели со списком БД щелкнуть правой кнопкой мыши, и в контекстном меню выбрать опцию **Create Schema** (рис. 13.1).

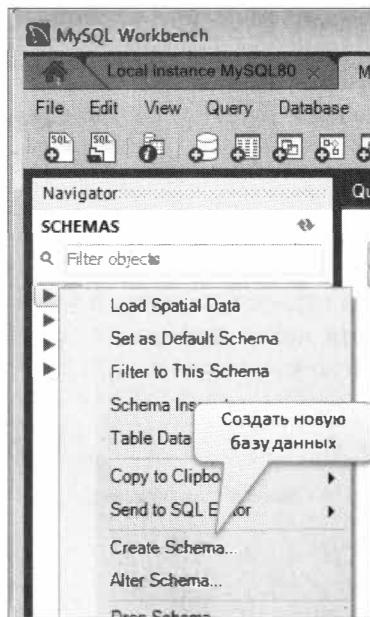


Рис. 13.1. Создание новой базы данных через панель **SCHEMAS**

Для того чтобы создать новую БД через панель инструментов, нужно в области этой панели щелкнуть левой кнопкой мыши на значке, обозначающем добавление новой БД (рис. 13.2).

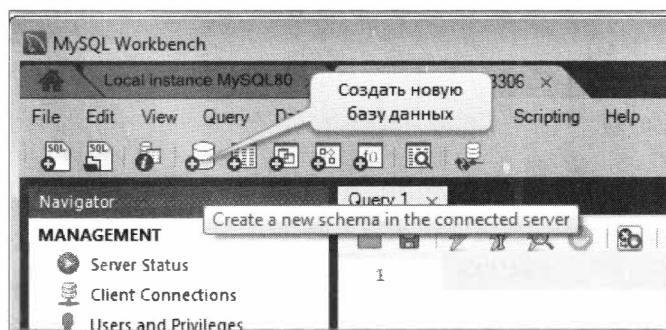


Рис. 13.2. Создание новой базы данных через панель инструментов

В обоих случаях после этих действий появится вкладка, в которой нужно ввести имя новой БД и указать параметры сортировки. В данном случае для примера база данных будет названа `book_db`. Параметры сортировки можно либо выбрать из выпадающего списка, либо оставить те, которые предлагаются по умолчанию. В нашем примере оставлены параметры по умолчанию (рис. 13.3).

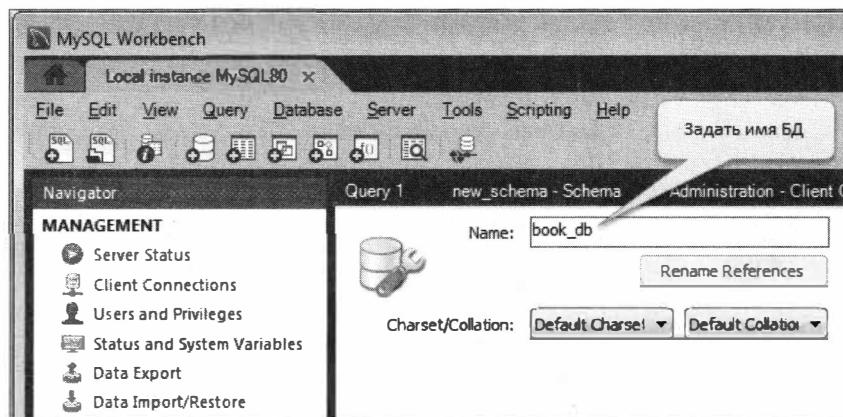


Рис. 13.3. Ввод имени новой базы данных

Затем нужно нажать на кнопку **Apply**, которая находится в нижней части этой вкладки. В результате откроется окно с текстом скрипта, который создаст новую БД. При необходимости скрипт можно отредактировать прямо в этом окне (рис. 13.4).

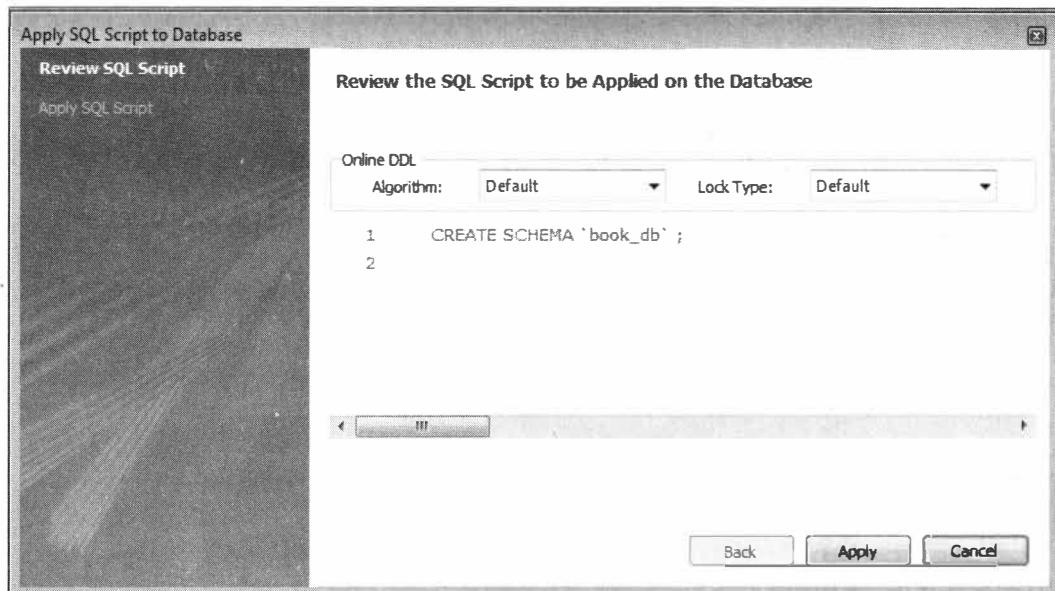


Рис. 13.4. Скрипт для создания новой БД

В верхней части окна имеется область **Online DDL**, предназначенная для установки параметров выполнения скрипта. Эти параметры могут быть полезны при манипуляциях с уже существующей БД. При создании новой БД рекомендуется оставить эти значения по умолчанию (**Default**). При нажатии на кнопку **Apply** откроется новое окно, в котором будет предложено выполнить данный скрипт (рис. 13.5).

После нажатия на кнопку **Finish** база данных будет создана. В левой панели на вкладке **SCHEMAS** появится информация о созданной базе данных (рис. 13.6).

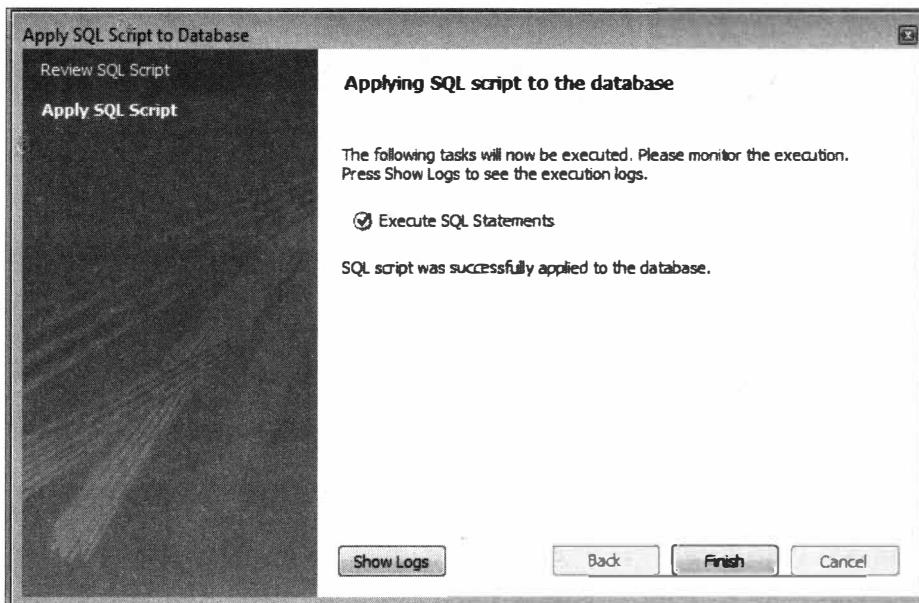


Рис. 13.5. Окно с предложением запуска скрипта на выполнение

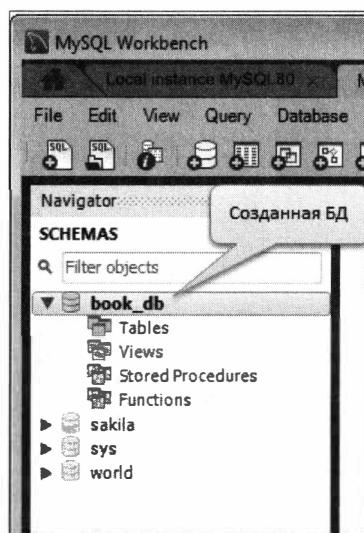


Рис. 13.6. Сведения о новой БД на вкладке SCHEMAS

Пока в данной базе нет таблиц, представлений, хранимых процедур и самих данных. Все это будем создавать позже средствами Django. Если в процессе работы над проектом потребуется подключение к нашей (или другой) БД, то это можно сделать из меню, выбрав опции **Database -> Connect to Database** (рис. 13.7).

После этого откроется окно, в котором нужно ввести имя базы данных и нажать на кнопку **OK** (рис. 13.8).

Итак, база данных создана, теперь можно перейти к проекту Django.

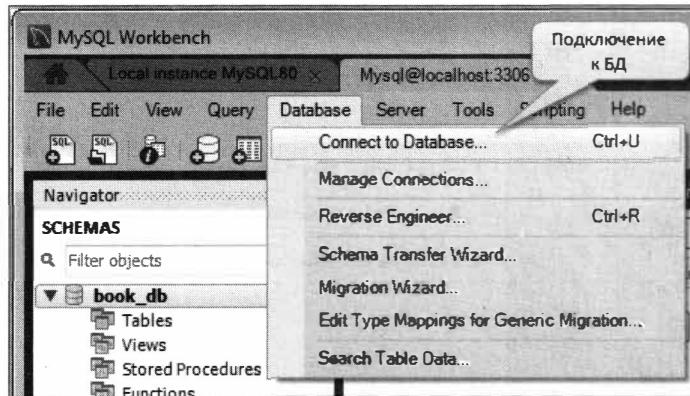


Рис. 13.7. Опции меню для открытия соединения с БД

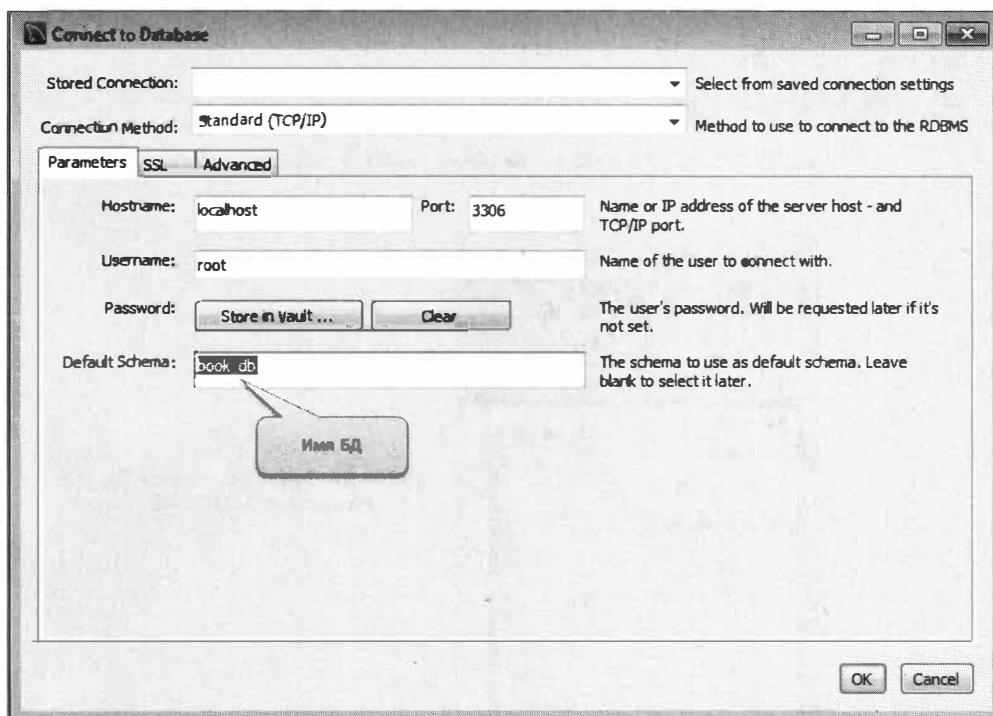


Рис. 13.8. Окно для открытия соединения с существующей БД

## 13.4. Создание проекта Django с базой данных MySQL на локальном компьютере

Займемся реализацией нового проекта с использованием СУБД MySQL. Откроем IDE PyCharm и создадим новый проект с именем Books\_SQL. Установим в данный проект все необходимые нам библиотеки и фреймворки, которые мы задействовали при создании сайта «Мир книг»:

```
pip install django;
pip install pillow;
pip install django-cleanup
```

Для взаимодействия с новой базой данных нам потребуется клиент MySQL, который можно загрузить с помощью следующей команды pip:

```
pip install mysqlclient
```

Теперь создаем проект Django с произвольным именем. Когда мы в предыдущих главах вели разработку веб-приложения, наш проект назывался WebBooks. Так как именно его мы будем переносить на удаленный сервер, то создадим проект Django с таким же именем. Выполним следующую команду:

```
django-admin startproject WebBooks
```

Перейдем в папку WebBooks:

```
cd WebBooks
```

Создадим приложение с именем catalog:

```
python manage.py startapp catalog
```

После этого нужно изменить настройки проекта, чтобы отключиться от БД SQLite и подключиться к нашей базе данных book\_sql.

Открываем файл settings.py, удаляем (или комментируем) строки настройки доступа к БД SQLite и вместо них добавляем код листинга 13.1.

### Листинг 13.1. Измененный код файла settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'book_db',
        'USER': 'root',
        'PASSWORD': '12344321',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

Во-первых, здесь мы заменили 'django.db.backends.sqlite3' на 'django.db.backends.mysql', указав тем самым, что мы переключились с базы данных SQLite на базу данных MySQL. Затем были заданы следующие параметры для работы с MySQL:

- 'NAME': book\_db — имя базы данных, к которой мы хотим подключиться;
- 'USER': root — имя пользователя MySQL — это тот, кто имеет доступ к базе данных и управляет ею (в данном случае главный администратор);
- 'PASSWORD': 12344321 — это пароль базы данных;
- 'HOST': localhost — локальный компьютер;
- 'PORT': 3306 — это число указывают, если база данных MySQL доступна по порту 3306.

В окне терминала PyCharm запускаем команды миграции:

```
python manage.py makemigrations
python manage.py migrate
```

После этого можно создать суперпользователя базы данных, выполнив в окне терминала PyCharm команду:

```
python manage.py createsuperuser
```

В процессе создания суперпользователя вводим пароль и электронный адрес.

Фактически мы в новом виртуальном окружении сформировали скелет будущего проекта. Открываем еще одно окно IDE PyCharm с проектом, в котором была использована СУБД SQLite. Теперь сравниваем содержимое файлов этих двух проектов и аккуратно переносим все файлы из старого проекта в новый.

После миграции в базе данных будут созданы все необходимые таблицы. Это можно проверить. Открываем MySQL Workbench и видим в ней таблицы нашего приложения catalog (рис. 13.9).

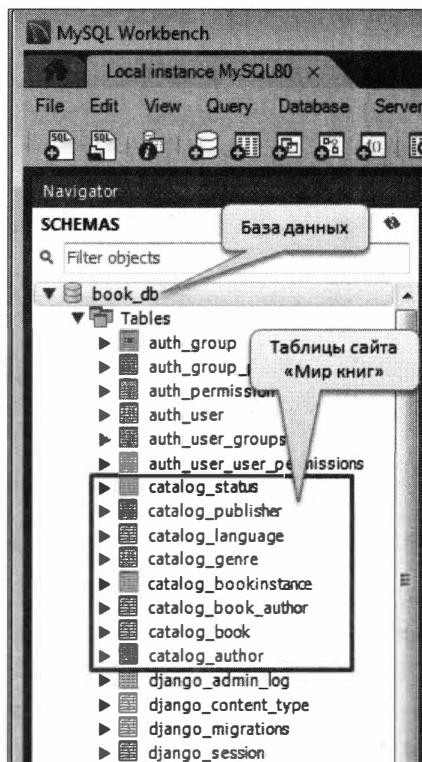


Рис. 13.9. Таблицы приложения catalog в БД MySQL

Все работы по переключению на СУБД MySQL выполнены. Можно запустить локальный сервер и проверить работу сайта. В окне терминала PyCharm запускаем сервер:

```
python manage.py runserver
```

Если все было сделано правильно, то должен открыться наш сайт (рис. 13.10).

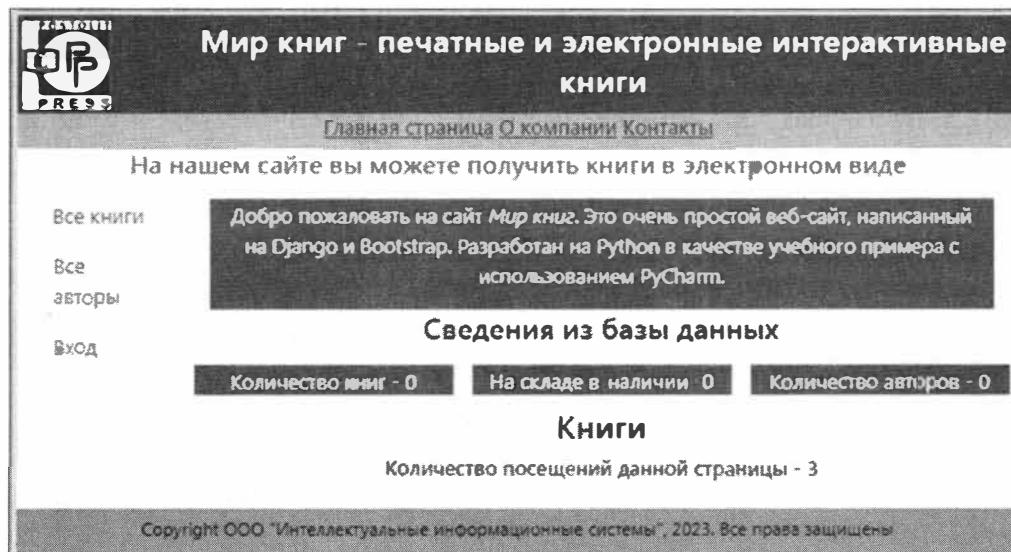


Рис. 13.10. Сайт «Мир книг» после переключения на СУБД MySQL

Как видно из рис. 13.10, база данных пустая. В ней нет сведений ни об авторах, ни о книгах, которые мы вносили ранее, т. к. в процессе миграции создаются только таблицы, а данные не переносятся. Теперь можно переносить сайт на хостинг уже с базой данных на MySQL.

## 13.5. Создание инфраструктуры на удаленном сервере для сайта с базой данных на MySQL

Для переноса сайта на хостинг с базой данных на MySQL нужно выполнить те же действия, которые были описаны в предыдущей главе, за исключением финальных операций, связанных с установкой и подключением к базе данных. Исходим из того, что у нас уже есть зарегистрированный аккаунт на хостинге timeweb.

Для создания нового сайта нужно выполнить несколько шагов:

- подключиться к серверу;
- создать папку для размещения файлов нашего сайта;
- создать виртуальное окружение для размещения нашего проекта на Django;
- установить в виртуальное окружение Python, Django и другие дополнительные библиотеки.

Для того чтобы не нарушить работу сайта, который был размещен на хостинге timeweb в предыдущей главе, мы создадим новый пустой сайт, в который будем переносить свое приложение на MySQL.

### ПРИМЕЧАНИЕ

Так как мы уже переносили сайт на хостинг timeweb в предыдущей главе, то некоторые несущественные окна при взаимодействии с удаленным сервером будут пропущены.

Для создания нового сайта открываем вкладку **Сайты** и выбираем опцию **Мои сайты**. После этого появится окно выбора типа создаваемого сайта. Поскольку наш сайт уже разработан, и нам его нужно просто перенести на публичный сервер, мы выбираем опцию **Создать новую директорию**. В открывшемся окне вводим имя нашего сайта «`newsite_sql`» и нажимаем кнопку **Создать**. После того как завершится процесс формирования пустой папки-шаблона, в списке созданных сайтов появится новый сайт с именем `newsite_sql` (рис. 13.11).

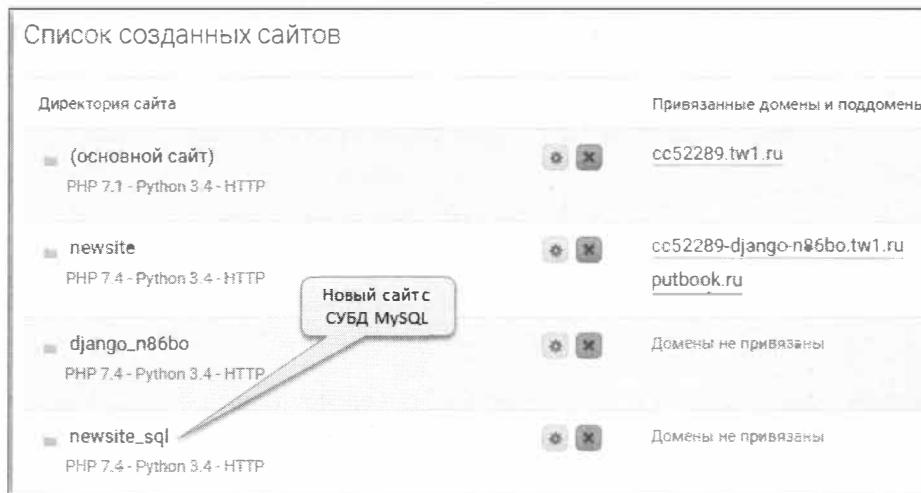


Рис. 13.11. Новый сайт в окне со списком созданных сайтов

Как видно из данного рисунка, у нового сайта нет привязанного доменного имени, а значит, мы не сможем проверить его работу через Интернет. Однако у нас есть свободные имена доменов второго уровня, которые можем подключить. Отвязываем доменное имя `cc52289-django-n86bo.tw1.ru` от сайта `newsite` и привязываем его к новому сайту `newsite_sql` (рис. 13.12).



Рис. 13.12. Привязка к сайту `newsite_sql` временного доменного имени

Проверим корректность переключения доменного имени. Копируем ссылку [cc52289-django-n86bo.tw1.ru](http://cc52289-django-n86bo.tw1.ru) и открываем этот сайт в новой вкладке браузера. Если домен переключился корректно, то будет показана главная страница сайта, созданная по умолчанию (рис. 13.13).

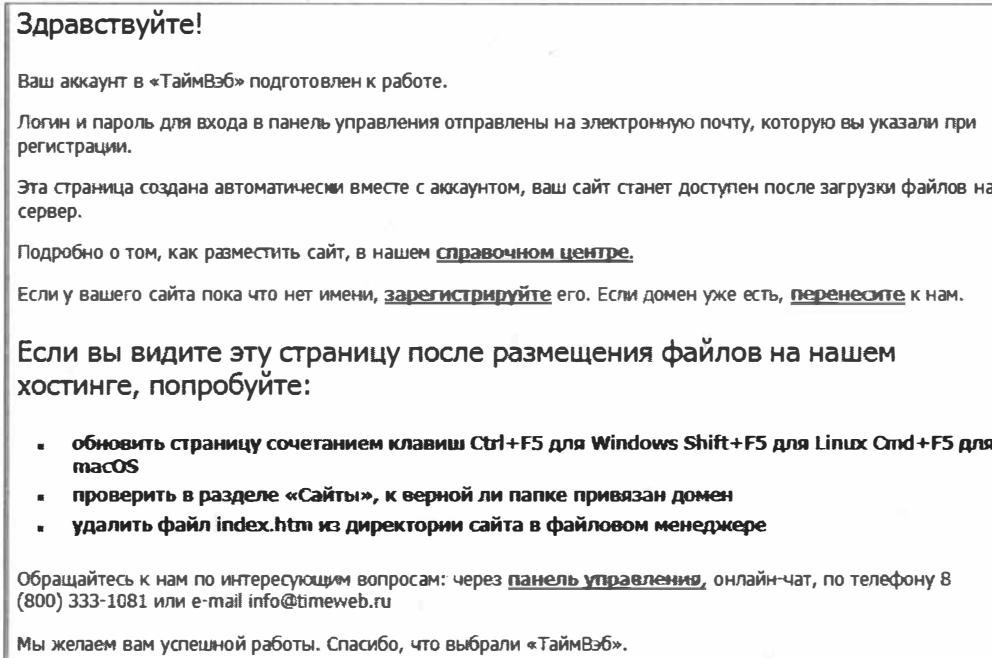


Рис. 13.13. Главная страница сайта newsite\_sql, созданная по умолчанию

На следующем этапе нужно сделать три шага:

- создать виртуальное окружение;
- установить в это виртуальное окружение необходимое программное обеспечение;
- создать проект Django.

Так как на удаленном сервере наше приложение будет работать под управлением Linux, то нам нужно подключиться к серверу через SSH-консоль. Если она у вас закрыта, необходимо снова ее открыть. После того как откроется консоль нужно войти в директорию newsite\_sql, для чего в окне терминала выполнить команду:

```
~$ cd newsite_sql
```

Далее скачиваем виртуальное окружение, указав в команде нужную версию Python:

```
~$ wget https://bootstrap.pypa.io/virtualenv/3.6/virtualenv.pyz
```

Проверяем версию Python:

```
python3 -V
```

Создаем виртуальное окружение для нашего проекта, выполнив следующую команду (здесь `venv` — имя папки для виртуального окружения):

```
~$ python3 virtualenv.pyz venv
```

Активируем виртуальное окружение, выполнив команду:

```
~$ source venv/bin/activate
```

Устанавливаем фреймворк Django:

```
~$ pip install django
```

Так как мы в своем проекте использовали еще две библиотеки (`pillow` и `django_cleanup`), то их тоже нужно установить:

```
~$ pip install pillow
~$ pip install django_cleanup
```

Эти библиотеки нам понадобятся для работы нашего приложения `catalog`.

Переходим в папку `public_html`:

```
~$ cd public_html
```

В этой папке создаем проект Django с произвольным именем. Когда мы вели разработку веб-приложения на локальном компьютере, наш проект назывался `WebBooks`. Так как именно его мы будем переносить, то на удаленном сервере создадим проект Django с таким же именем. Выполним команду

```
~$ django-admin.py startproject WebBooks
```

На данном этапе мы добились следующих результатов:

- создали виртуальное окружение;
- загрузили в виртуальное окружение необходимый софт: Python, Django, pillow, `django_cleanup`;
- создали проект Django с именем `WebBooks`.

Теперь можно закрыть SSH-консоль и вернуться в веб-панель личного кабинета `timeweb`. Здесь можно посмотреть структуру папок нашего сайта с использованием файлового менеджера (рис. 13.14).

На следующем шаге через файловый менеджер переходим в папку `newsite/public_html` и создаем там новый файл с именем `.htaccess`. Сразу создать файл с таким именем не



Рис. 13.14. Проект Django с именем `WebBooks`, созданный на удаленном сервере

получится, т. к. к любому созданному файлу автоматически добавляется расширение. Поэтому сначала создаем файл с именем `.htaccess.html`, а потом переименовываем его, задав новое имя `.htaccess`. Обратите внимание — имя файла начинается с точки.

Что должно содержаться в данном файле, мы можем подсмотреть в папке с сайтом, который был создан в предыдущей главе (в нашем случае это сайт с именем `newsite`). Переходим на данный сайт, заходим в папку `newsite/public_html`, открываем файл с именем `.htaccess`. Он будет содержать следующий код (рис. 13.15).

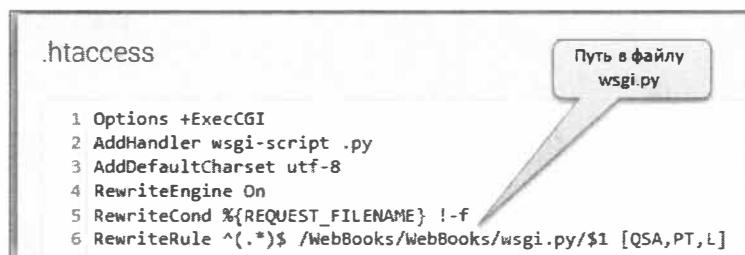


Рис. 13.15. Код файла `.htaccess` сайта Django

В этом файле нужно указать путь к модулю `wsgi.py` для того, чтобы перенаправлять запросы пользователя к этому модулю. Копируем код этого файла в буфер и затем вставляем его в аналогичный файл нашего проекта. Теперь файл `.htaccess` на сайте `newsite_sql` будет содержать код листинга 13.2.

### Листинг 13.2. Код файла `.htaccess`

```

Options +ExecCGI
AddHandler wsgi-script .py
AddDefaultCharset utf-8
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ /WebBooks/WebBooks/wsgi.py/$1 [QSA,PT,L]

```

Здесь мы фактически указали путь к файлу `wsgi.py`. Именно по этому пути он находится в папках нашего проекта `WebBooks`. Нажимаем кнопку **Сохранить** и закрываем данное окно. В файловом менеджере можно проверить, что файл `wsgi.py` находится именно по этому пути (рис. 13.16).

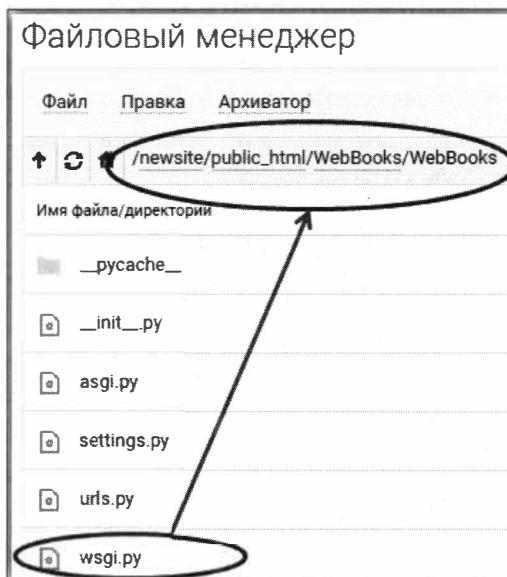
Затем в этой же папке находим файл `settings.py`, открываем его и вносим минимальные корректировки (листинг 13.3, изменения выделены серым фоном).

### Листинг 13.3. Измененный код файла `settings.py`

```

import os
from pathlib import Path
...
LANGUAGE_CODE = 'ru-ru'
TIME_ZONE = 'Europe/Moscow'

```



**Рис. 13.16.** Место расположения файла `wsgi.py`

Здесь мы добавили импорт пакета `os`, изменили язык интерфейса административной панели Django на русский, сменили часовой пояс для указания текущего времени.

Находим строку `ALLOWED_HOSTS = []` и меняем ее, как показано в листинге 13.4 (изменения выделены серым фоном).

#### Листинг 13.4. Измененный код файла `settings.py`

```
ALLOWED_HOSTS = [/**/*]
```

Здесь необходимо указать доменное имя (или список допустимых доменных имен). На период отладки мы вместо доменного имени поставили звездочку, это означает, что данный сайт допускает использование любых доменных имен. Когда реальное доменное имя будет получено и зарегистрировано, этот параметр можно будет поменять.

Нам осталось отредактировать содержимое файла `wsgi.py`, который находится в этой же папке. Что должно содержаться в данном файле, мы снова можем подсмотреть в сайте Django, который был создан в предыдущей главе (в нашем случае это сайт с именем `newsite`). Переходим на данный сайт, заходим в папку `newsite/public_html/WebBooks/WebBooks/`, в которой открываем файл с именем `wsgi.py`. Он будет содержать код листинга 13.5.

#### Листинг 13.5. Код файла `wsgi.py` сайта `newsite`

```
# -*- coding: utf-8 -*-
import os
import sys
import platform
# путь в проекте к модулю manage.py
sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks')
# путь в проекте к модулю settings.py
sys.path.insert(0, '/home/c/cc52289/newsite/public_html/WebBooks/')
```

```
# путь к виртуальному окружению
sys.path.insert(0, '/home/c/cc52289/newsite/venv/lib/python{0}/site-
packages'.format(platform.python_version()[0:3]))
os.environ["DJANGO_SETTINGS_MODULE"] = "WebBooks.settings"
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

В этом файле нужно будет изменить три строки (выделено серым фоном). Копируем код файла в буфер. На нашем сайте newsite\_sql открываем аналогичный файл wsgi.py, удаляем его содержимое и вставляем код из буфера. Теперь корректируем этот файл, как показано в листинге 13.6 (изменения выделены серым фоном).

#### Листинг 13.6. Код файла wsgi.py сайта newsite\_sql

```
# -*- coding: utf-8 -*-
import os
import sys
import platform
# путь в проекте к модулю manage.py
sys.path.insert(0, '/home/c/cc52289/newsite_sql/public_html/WebBooks')
# путь в проекте к модулю settings.py
sys.path.insert(0, '/home/c/cc52289/newsite_sql/public_html/WebBooks/WebBooks')
# путь к виртуальному окружению
sys.path.insert(0, '/home/c/cc52289/newsite_sql/venv/lib/python{0}/site-
packages'.format(platform.python_version()[0:3]))
os.environ["DJANGO_SETTINGS_MODULE"] = "WebBooks.settings"
from django.core.wsgi import get_wsgi_application
application = get_wsgi_application()
```

В данном файле мы указали реальные пути на удаленном сервере к следующим файлам:

- путь к модулю `manage.py`;
- к модулю `settings.py`;
- к виртуальному окружению (`venv`);
- к параметрам настройки.

Остался один шаг — проверить работу установленного приложения Django. Если нажать левой кнопкой мыши на временное доменное имя (в нашем случае это **cc52289-django-n86bo.tw1.ru**) или открыть эту ссылку в новой вкладке браузера, то должна загрузиться административная панель Django (рис. 13.17).

Если вы увидели подобную картину, значит, все действия на предыдущих этапах выполнены правильно.

Теперь в проекте Django с именем WebBooks создадим приложение. Так как в нашем проекте на локальном компьютере приложение называлось catalog, то оставим это имя без изменения. Снова открываем SSH-консоль, переходим в папку с нашим сайтом и проектом `/newsite_sql/public_html/WebBooks`:

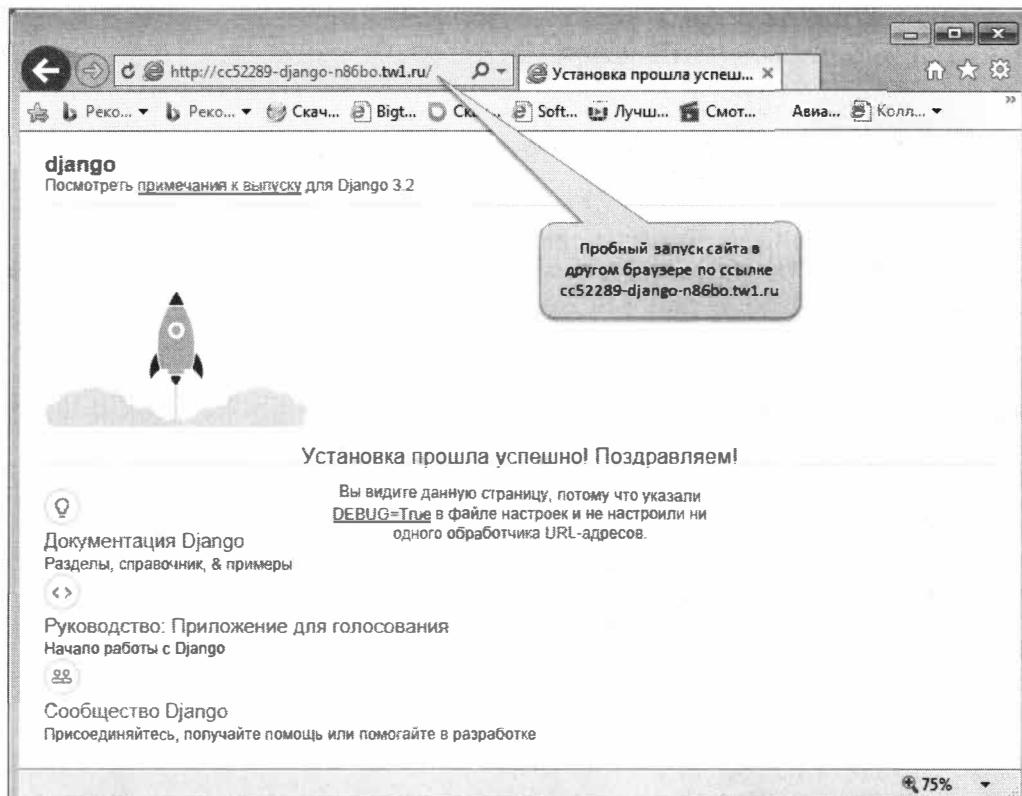


Рис. 13.17. Административная панель Django при активации временного доменного имени

```
cd newsite_sql
cd public_html
cd WebBooks
```

Вот теперь, находясь в папке проекта Django с именем **WebBooks** (той, где содержится скрипт `manage.py`), создадим приложение `catalog`. Для этого выполним следующую команду:

```
python manage.py startapp catalog
```

После этого в проекте появится новая папка с именем `catalog`, в которой будут находиться все модули этого приложения (рис. 13.18).

Этот проект теперь нужно добавить в список установленных приложений, как и модуль `django_cleanup`, который у нас используется для очистки хранилища медиафайлов. Для этого открываем файл `settings.py` и в раздел `INSTALLED_APPS` добавляем две строки (листинг 13.7, изменения выделены серым фоном)

#### Листинг 13.7. Измененный код файла `settings.py`

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
```

```
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'catalog',
'django_cleanup',  
]
```

Теперь можно перейти к следующему этапу — созданию базы данных MySQL.

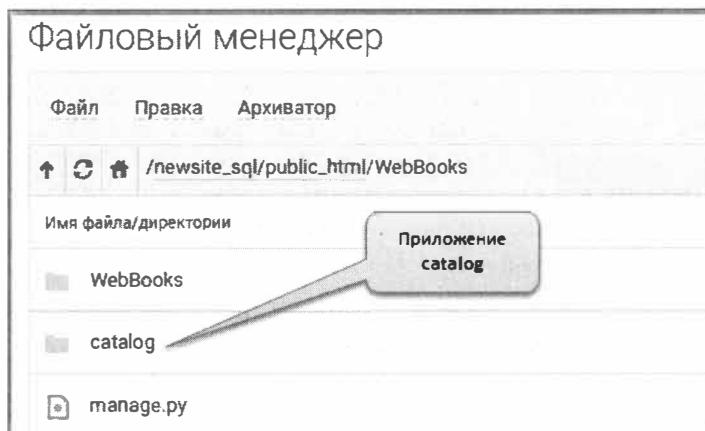


Рис. 13.18. Папка с приложением Django catalog

## 13.6. Создание базы данных MySQL на удаленном сервере

Когда мы вели разработку проекта на локальном компьютере, нам пришлось устанавливать сервер MySQL и создавать в нем новую базу данных. На хостинге timeweb можно пойти тем же путем — создать новую базу данных. Для этого нужно в левой панели инструментов с помощью левой кнопки мыши выбрать опцию **Базы данных**. После этого откроется вкладка для работы с базами данных, в которой есть опция **Создание новой базы данных** (рис. 13.19).

Как видно из рис. 13.19, на панели управления есть опция **Создание новой базы данных**. Здесь с использованием соответствующих инструкций можно создать новую БД. Но т. к. мы ранее инсталлировали проект Django на основе CMS, то в нашем распоряжении имеется уже установленная база данных MySQL (cc52289\_n86bo), к которой мы можем подключиться. Воспользуемся этой возможностью.

Тип подключения к базе данных будет зависеть от того, на каких серверах находится эта БД и на каком сервере размещен сайт, который ее использует. Если и база данных, и сайт находятся на одном сервере, такое подключение называют локальным. Если же база данных и сайт расположены на разных серверах, то устанавливается удаленное подключение.

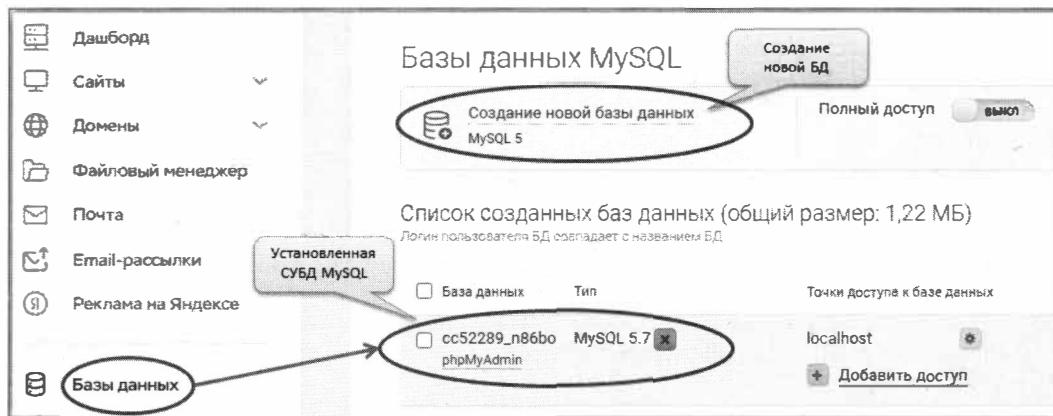


Рис. 13.19. Вкладка Базы данных MySQL в панели управления хостинга timeweb

У нас и сайт, и база данных находятся на одном сервере. Для локального подключения к базе данных потребуются следующие реквизиты:

- сервер — localhost (или 127.0.0.1);
- имя базы данных — оно отображено на панели управления **Базы данных**;
- имя пользователя — совпадает с именем базы данных;
- пароль — формируется при создании базы данных.

Пароль можно посмотреть в конфигурационном файле `settings.ru` сайта, который был создан на основе CMS, либо изменить в разделе **Базы данных**, щелкнув мышью на значке шестеренки в строке с именем БД (рис. 13.20).

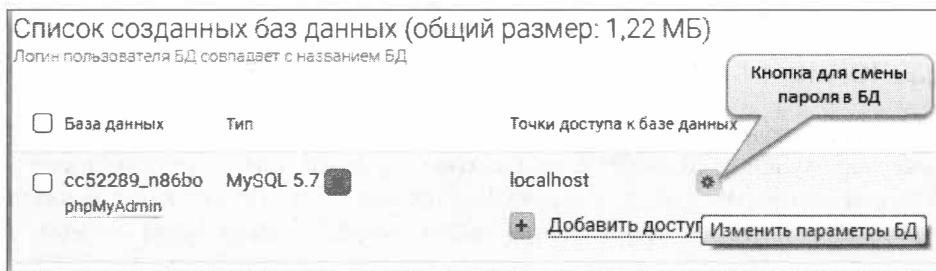


Рис. 13.20. Кнопка для активации функции смены пароля

Мы не будем создавать новую БД, а воспользуемся той, которая установлена по умолчанию (`cc52289_n86bo`). Щелкнем левой кнопкой мыши на значке с шестеренкой и в открывшемся окне поменяем пароль (рис. 13.21).

Теперь мы можем подключиться к базе данных. Для этого нужно щелкнуть левой кнопкой мыши по ссылке, которая находится под именем БД в веб-консоли в панели управления (рис. 13.22).

После этого появится окно для ввода пароля доступа к БД (рис. 13.23):

После ввода пароля и нажатия на кнопку **Войти** в новой вкладке браузера откроется административная панель для работы с базой данных (рис. 13.24).

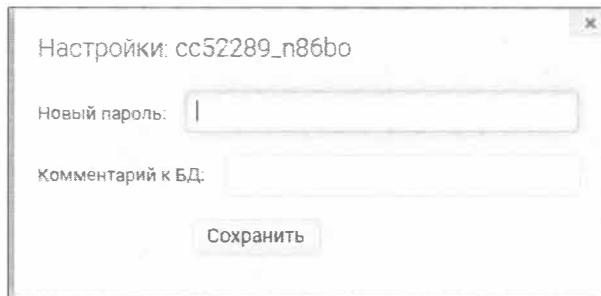


Рис. 13.21. Окно для смены пароля

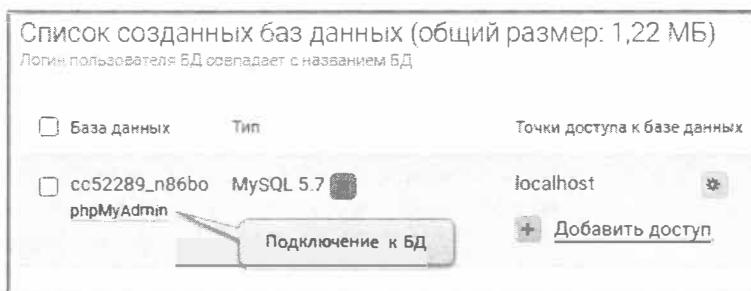


Рис. 13.22. Ссылка на подключение к базе данных на сервере

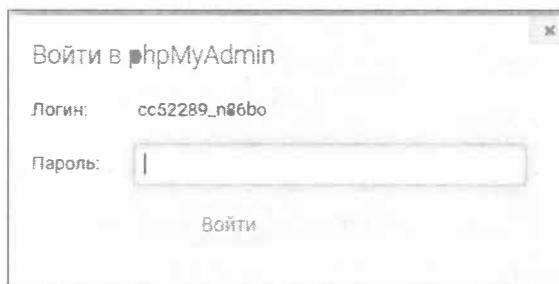


Рис. 13.23. Окно для ввода пароля

Как видно из рис. 13.24, на сервере localhost создана база данных cc55289\_n86bo.

Для подключения к этой БД можно задать следующие параметры:

- имя сервера — localhost;
- имя базы данных — cc55289\_n86bo;
- имя пользователя — cc55289\_n86bo;
- пароль — указывался при смене пароля.

Мы можем подсмотреть параметры подключения к данной БД на сайте django\_n86bo, который был создан на основе CMS. В файловом менеджере открываем данный сайт и находим там файл settings.py. В коде этого файла находим словарь DATABASES (рис. 13.25).

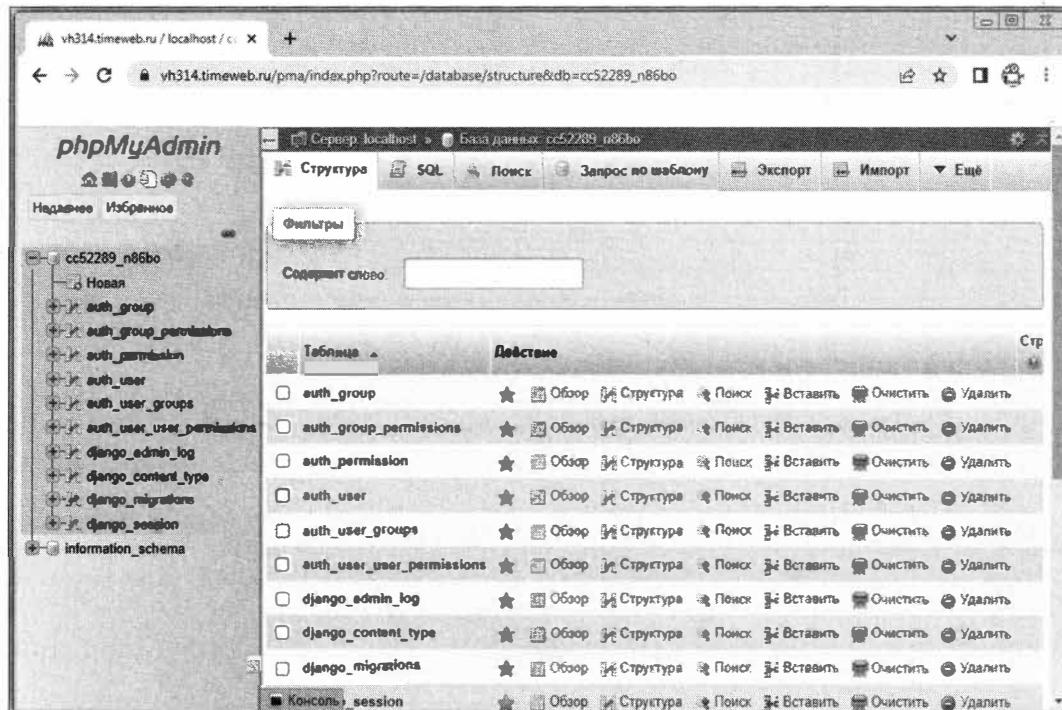
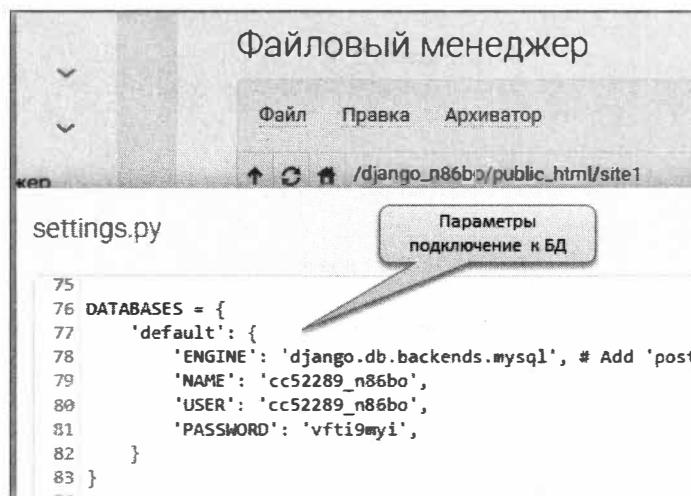


Рис. 13.24. Административная панель для работы с базой данных



Копируем в буфер обмена этот код. Затем открываем файл `settings.py` нашего нового сайта `newsite_sql` и добавляем код листинга 13.8 (изменения выделены серым фоном).

**Листинг 13.8. Измененный код файла `settings.py`**

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
}  
  
#  
  
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql', # Add 'mysql'  
        'NAME': 'cc52289_n86bo',  
        'USER': 'cc52289_n86bo',  
        'PASSWORD': 'здесь_ваш_пароль',  
    }  
}
```

Здесь мы временно закомментируем строки подключения к базе данных MySQL и оставим подключение к БД SQLite (это нам потребуется для проверки корректности работы сайта после переноса основных файлов проекта).

## 13.7. Перенос сайта с локального компьютера на публичный сервер

Итак, теперь можно приступить к переносу веб-приложения с локального рабочего компьютера на публичный сервер. Для переноса данных нам понадобится файловый менеджер, который позволяет взаимодействовать с удаленными компьютерами. Мы воспользуемся файловым менеджером Total Commander.

Параметры подключения к удаленному серверу по протоколу FTP можно найти в панели управления на вкладке **Дашборд** в разделе **Доступ по FTP** (рис. 13.26).

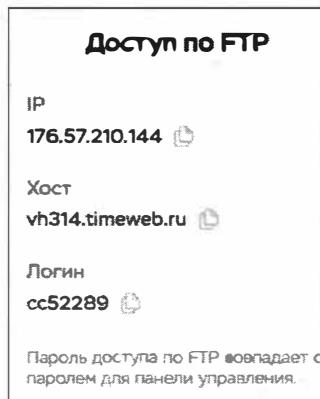


Рис. 13.26. Параметры FTP-доступа к удаленному серверу

Открываем файловый менеджер и в меню **Сеть** выбираем опцию **Новое FTP-соединение** (рис. 13.27).

В открывшемся окне вводим параметры удаленного сервера и нажимаем кнопку **OK** (рис. 13.28).

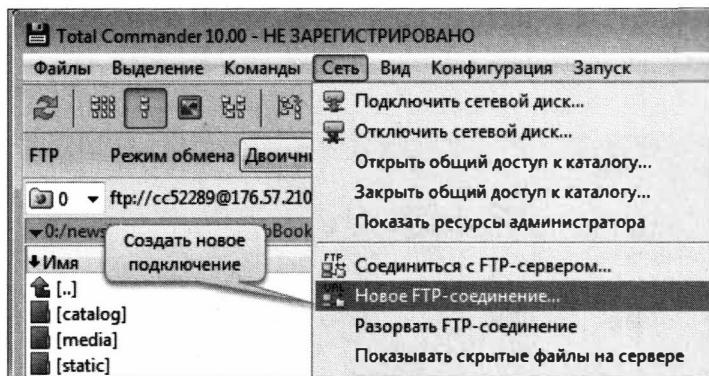


Рис. 13.27. Создание нового соединения в окне файлового менеджера

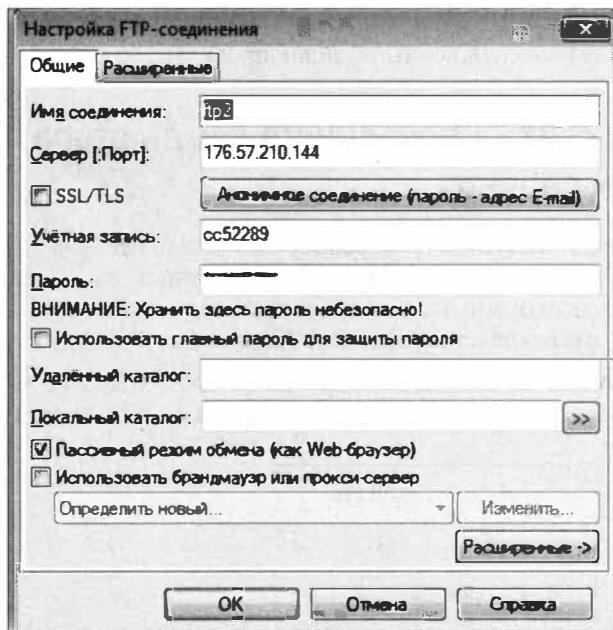


Рис. 13.28. Открытие подключения к удаленному серверу

После этого откроется окно, в котором будут две панели (рис. 13.29).

Как видно из данного рисунка, в левой панели открыты папки с нашим проектом WebBooks на удаленном сервере, а в правой панели открываем папку с проектом WebBooks на локальном компьютере, где велась разработка проекта.

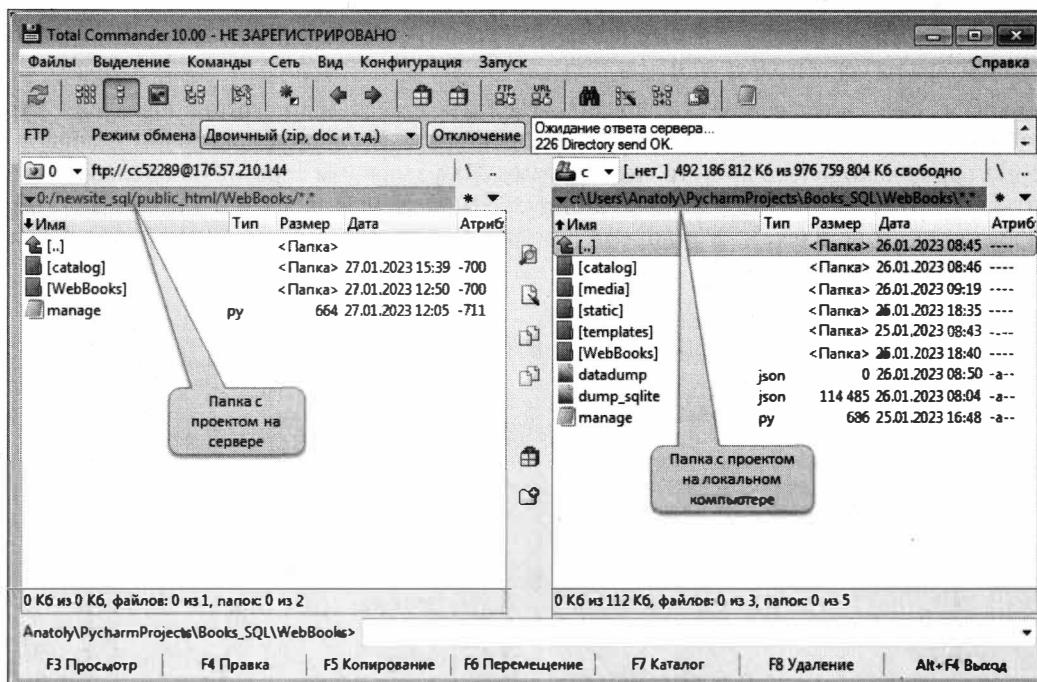


Рис. 13.29. Окно файлового менеджера Total Commander

Теперь с помощью файлового менеджера копируем на удаленный сервер следующие файлы из папки `catalog` (там находится наше приложение) в папку `catalog` на удаленном сервере:

- `admin.py`;
- `forms.py`;
- `models.py`;
- `urls.py`;
- `views.py`.

В папку `public_html/WebBooks/` удаленного сервера копируем следующие папки с локального компьютера:

- `media` — в этой папке находятся медиафайлы, которые пользователи могут загружать удаленно (изображения и документы);
- `static` — файлы с изображениями, скрипты и CSS-файлы (в том числе файлы Bootstrap);
- `templates` — шаблоны HTML-страниц.

Для проверки корректности работы сайта можно перенести и файл `db` (база данных SQLite3). Этот файл можно взять с локального компьютера или с сервера с уже развернутого сайта `newsite`.

Теперь и в левой, и в правой панели открываем папку `WebBooks/WebBooks`. В нашем проекте в данной папке находится файл `urls.py`. Переносим этот файл с локального компью-

тера на удаленный сервер. На этом процессе переноса файлов нашего сайта с локального компьютера на сервер завершен.

Теперь нужно снова вернуться к файловому менеджеру личного кабинета на сайте timeweb и перенести папку static в другую директорию. На нашем локальном компьютере эта папка находилась по пути WebBooks/static. На удаленном сервере она должна находиться в папке newsite/public\_html/static. Копируем папку static в директории newsite/public\_html/WebBooks/static и вставляем ее в директорию newsite/public\_html/static. Это изменение продемонстрировано на рис. 13.30.

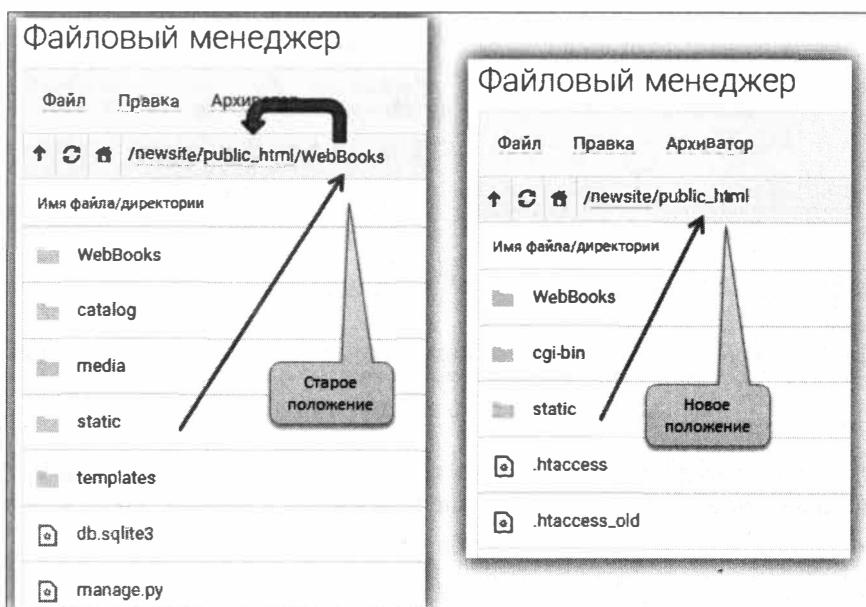


Рис. 13.30. Изменение расположения папки static

Как видно из данного рисунка, папку static нужно поднять на один уровень вверх (все вложенные папки с их содержимым тоже должны переместиться).

Осталось внести некоторые изменения в файл WebBooks/WebBooks/settings.py. Открываем данный файл на удаленном сервере и меняем строки листинга 13.9 (изменения выделены серым фоном).

#### Листинг 13.9. Измененный код файла settings.py

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

Здесь мы в параметр `INSTALLED_APPS` добавили наше приложение (`catalog`) и пакет, предназначенный для очистки хранилища медиафайлов (`django_cleanup`).

Далее добавляем строки, указывающие место расположения шаблонов HTML-страниц (листинг 13.10, изменения выделены серым фоном).

#### Листинг 13.10. Измененный код файла `settings.py`

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

```

Указываем новые пути к папке `static` и медиафайлам (листинг 13.11, изменения выделены серым фоном).

#### Листинг 13.11. Измененный код файла `settings.py`

```
STATIC_URL = '/static/'
STATIC_ROOT = '/home/c/cc52289/newsite/public_html/static'
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media/')
```

В конце файла добавляем строки листинга 13.12 (изменения выделены серым фоном).

#### Листинг 13.12. Измененный код файла `settings.py`

```
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
# Переадресация на главную страницу сайта после входа в систему
LOGIN_REDIRECT_URL = '/'
# настройки отправки e-mail
''' это пробная отправка на консоль '''
# EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
''' это реальная отправка через smtp.gmail '''
EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
EMAIL_HOST = 'smtp.gmail.com'
# EMAIL_USE_TLS = True
EMAIL_USE_SSL = True # для отправки с gmail
```

```
EMAIL_PORT = 465
EMAIL_HOST_USER = "ваша_почта@gmail.com" # от кого
EMAIL_HOST_PASSWORD = "ваш_пароль" # пароль почты отправителя
```

Здесь указан параметр для переадресации на главную страницу (`LOGIN_REDIRECT_URL = '/'`), и параметры отправки электронной почты внешним пользователям для смены пароля.

### **ПРИМЕЧАНИЕ**

Параметры для отправки электронной почты в вашем приложении будут другими, привязанными к вашему адресу и вашему паролю.

После всех этих изменений наш сайт должен быть доступен в Интернете по временному доменному имени (в нашем случае это `cc52289-django-n86bo.tw1.ru`). Для чистоты эксперимента копируем эту ссылку и вставляем ее в адресную строку интернет-браузера. Если никаких ошибок не было, то должна открыться главная страница нашего сайта (рис. 13.31).

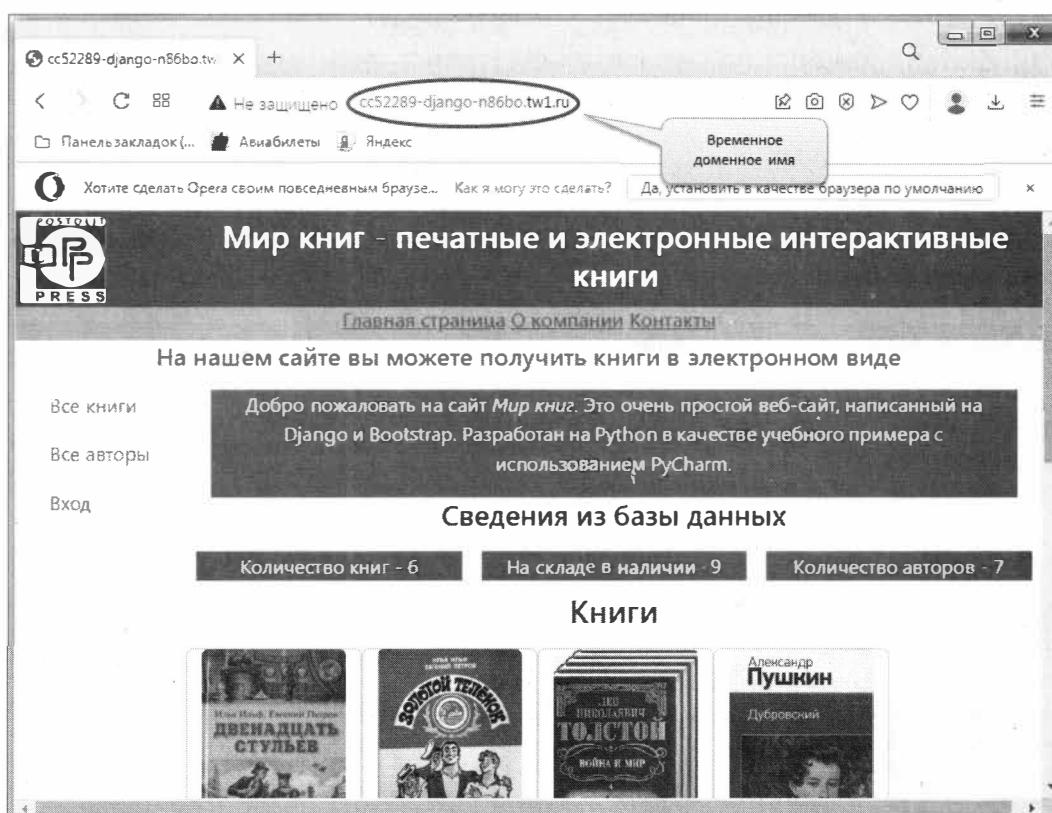


Рис. 13.31. Главная страница сайта, загруженная с удаленного сервера

Если ваша страница загрузилась, значит, все настройки в процессе переноса были выполнены правильно. Но это был тестовый запуск с подключением к БД на SQLite. Теперь пора переключить наш сайт на СУБД MySQL.

Снова в файловом менеджере открываем файл `settings.py`. Теперь закомментируем строки подключения к базе SQLite (выделено серым фоном, впоследствии их можно удалить) и снимаем комментарии со строк подключения к базе данных MySQL (листинг 13.13).

#### Листинг 13.13. Измененный код файла `settings.py`

```
'''  
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': BASE_DIR / 'db.sqlite3',  
    }  
    ...  
}  
  
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.mysql', # Add 'mysql'  
        'NAME': 'cc52289_n86bo',  
        'USER': 'cc52289_n86bo',  
        'PASSWORD': 'здесь_ваш_пароль',  
    }  
}
```

После изменения параметров подключения можно выполнить миграцию таблиц в базу данных MySQL. Открываем SSH-консоль и переходим в папку с нашим сайтом `newsite_sql`:

```
~$ cd newsite_sql
```

Активируем виртуальное окружение, подав команду:

```
~$ source venv/bin/activate
```

Устанавливаем MySQL-клиент:

```
~$ pip install mysqlclient
```

Переходим в папку `/newsite_sql/public_html/WebBooks`:

```
~$ cd public_html
```

```
~$ cd WebBooks
```

Выполняем миграцию таблиц базы данных:

```
~$ python3 manage.py makemigrations
```

```
~$ python3 manage.py migrate
```

После миграции на удаленном сервере в БД My\_SQL буду созданы таблицы приложения `catalog`. В этом можно убедиться, если открыть административную панель для работы с базами данных (рис. 13.32).

Если теперь снова открыть сайт с использованием временного доменного имени, то мы увидим главную страницу сайта, в которую загружены данные уже из БД My\_SQL (рис. 13.33).

Как видно из рис. 13.33, база данных пустая, т. к. в процессе миграции в нее переносятся только таблицы.

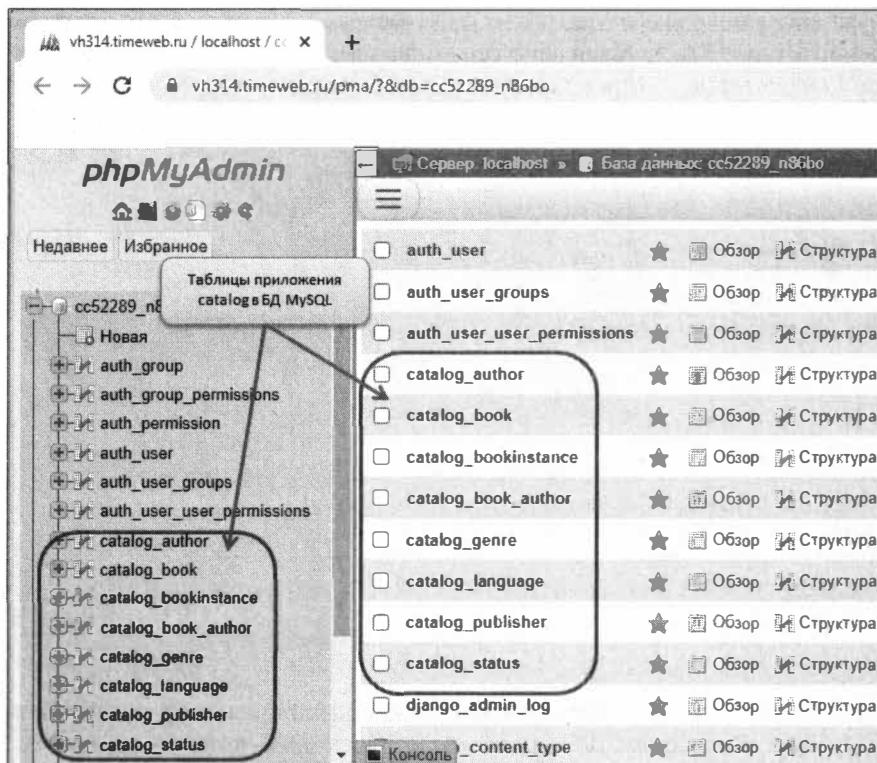


Рис. 13.32. Таблицы приложения catalog после миграции таблиц базы данных

cc52289-django-n86bo.tw1.ru

Не защищено cc52289-django-n86bo.tw1.ru

Панель закладок ... Авиабилеты Яндекс

Временное доменное имя

Хотите сделать Opera своим повседневным браузером? Как я могу это сделать? Да, установить в качестве браузера по умолчанию

**Мир книг - печатные и электронные интерактивные КНИГИ**

Главная страница Компания Контакты

На нашем сайте вы можете получить книги в электронном виде

Все книги Добро пожаловать на сайт Mir книг. Это очень простой веб-сайт, написанный на Django и Bootstrap. Разработан на Python в качестве учебного примера с использованием PyCharm.

Все авторы

Вход

Сведения из базы данных

Количество книг - 0      На складе в наличии - 0      Количество авторов - 0

Книги

Количество посещений данной страницы - 1

Рис. 13.33. Главная страница сайта «Мир книг» с информацией из БД My\_SQL

Теперь создадим суперпользователя, чтобы реализовать возможности удаленного администрирования. Возвращаемся к SSH-терминалу и вводим команду

```
python manage.py createsuperuser
```

При создании суперпользователя потребуется ввод его имени, электронного адреса и пароля. Если теперь войти на сайт с паролем суперпользователя, то будет загружена административная панель Django (рис. 13.34).

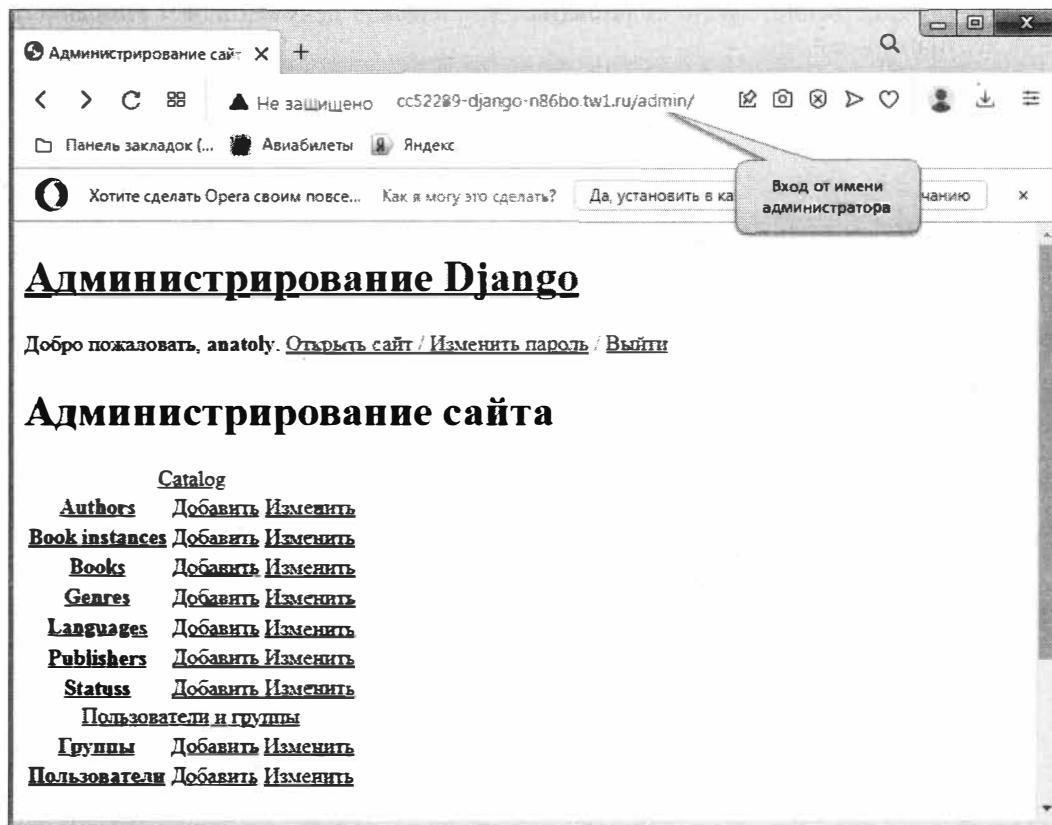


Рис. 13.34. Административная панель Django на сайте, развернутом на СУБД MySQL

Мы выполнили все шаги по разработке приложения Django на локальном компьютере и по переносу сайта на публичный сервер с базой данных MySQL. Теперь можно зарегистрировать доменное имя, сделать привязку сайта к этому доменному имени и заполнять его данными либо через интерфейс сайта, либо через административную панель.

## 13.8. Краткие итоги

В этой главе были рассмотрены вопросы разработки приложения Django с использованием СУБД MySQL и публикации готового веб-сайта в сети Интернет. Кому-то может показаться, что материалы данной главы несколько избыточны, поскольку они частично повторяют содержание предыдущих глав. Но здесь сознательно сделан повтор всех

шагов от создания приложения с применением MySQL на локальном компьютере, до его развертывания в Интернете. В этом случае при изучении материала нет необходимости постоянно возвращаться к разделам предыдущих глав.

Следует иметь в виду, что настройки доступа к сайту на хостинге timeweb, которые использовались при написании книги, со временем могут измениться. При возникновении нештатных ситуаций всегда можно обратиться к актуальной документации, размещенной на сайте [timeweb.ru](http://timeweb.ru). Если вы решите развернуть сайт на других серверных площадках, то, естественно, нужно пользоваться технической документацией выбранного вами хостинг-провайдера.

На этом мы заканчиваем знакомство с фреймворком Django. Удачных проектов и успехов в работе!

# Послесловие

В этой книге мы рассмотрели только базовые приемы разработки веб-приложений, обеспечивающие создание основных элементов сайтов. Приводимые в ней примеры и шаблоны HTML-страниц были приведены для лучшего их понимания и сознательно максимально упрощены. Основная цель книги — познакомить ее читателей со структурой приложений на Django, принципами взаимодействия между элементами этого фреймворка, дать вам возможность понять сущность методов доступа к базам данных. Ведь Django позволяет обращаться к таблицам СУБД не напрямую через SQL-запросы, а в терминах класса. Работая с Django, можно создать класс со своим набором переменных и методов, которые потом будут переписаны в SQL автоматически. При этом необходимость самому писать SQL-код отпадает, хотя и не исключена полностью. Если после знакомства с приведенными в книге материалами вы почувствовали, что Python и Django прекрасно работают в среде PyCharm и что применение этого набора инструментов упрощает программистам разработку веб-приложений, то основная цель книги достигнута.

Следующий шаг в освоении веб-технологий — детальное изучение инструментария для разработки и дизайна HTML-страниц и применения CSS-стилей, служащих основой для создания привлекательного пользовательского интерфейса. Здесь можно воспользоваться возможностями фреймворка Bootstrap. В книге о нем приведены только базовые сведения. Более детально возможности Bootstrap можно изучить в специальной литературе или в официальной документации, которая доступна в Интернете. Кроме того, в книге охвачены далеко не все тонкости разработки веб-приложений. Достаточно актуальны вопросы создания географических веб-приложений. У Django есть прекрасное расширение — GeoDjango, а также библиотека Mapnik и геопространственное расширение PostGIS для СУБД PostgreSQL. Набор этих модулей позволяет максимально упростить создание географических веб-приложений и служб, основанных на местоположении. Если это издание окажется востребованным, то следующим шагом автора будет подготовка книги по технологическим приемам программирования геопространственных веб-приложений.

А в завершение остается пожелать вам удачного применения в своей практической деятельности тех навыков и знаний, которые вы приобрели, прочитав эту книгу.

д.т.н., профессор *Anatolij Postolit*

# Список источников и литературы

1. CSS — сделай это красиво. <https://tutorial.djangogirls.org/ru/css/>.
2. CSS-шаблоны сайтов с использованием бестабличной верстки.  
<http://ruseller.com/adds.php?rub=36>.
3. DjangoBook. <https://djbook.ru/index.html>.
4. Django 4.1. Работа с формами.  
<https://runebook.dev/ru/docs/django/topics/forms/index>.
5. Django documentation. <https://django.readthedocs.io/en/stable/#django-documentation>.
6. django-embed-video 1.4.8. <https://pypi.org/project/django-embed-video/>.
7. Python. Урок 1. Установка. <https://devpractice.ru/python-lesson-1-install/>.
8. Python. Урок 16. Установка пакетов в Python.  
<https://devpractice.ru/python-lesson-16-install-packages/>.
9. Python. Загрузите последнюю версию для Windows.  
<https://www.python.org/downloads/>.
10. Web-программирование. Конспект лекций.  
<http://edu.cassiopeia.com.ua/lib/conspectHTML.pdf>
11. Антонио Меле. Django 2 в примерах. М.: Издательство ДМК, 2019.
12. Бесплатный django хостинг. <https://uzverss.livejournal.com/69130.html>.
13. Ваша первая модель. <https://djbook.ru/ch05s06.html>.
14. Веб-платформа Django (Python).  
<https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django>.
15. Васильева И. Н., Федоров Д. Ю. Web-технологии: Изд-во Санкт-Петербургского государственного экономического университета, 2014. — 67 с.  
[https://dfedorov.spb.ru/Web\\_Tehnologii.pdf](https://dfedorov.spb.ru/Web_Tehnologii.pdf).
16. Верстайте адаптивный дизайн быстро и легко. <https://bootstrap-v5.ru/>.
17. Виртуальный хостинг на timeweb. <https://timeweb.com/ru/>.
18. Встроенные Field классы.  
<https://runebook.dev/ru/docs/django/ref/forms/fields?page=2>.

19. Дронов В. А. Django 3.0. Практика создания веб-сайтов на Python. СПб.: БХВ-Петербург, 2021. — 704 с.
20. Документация Django 1.9. Поля формы.  
[https://ejudge.lksh.ru/lang\\_docs/djbook.ru/rel1.9/ref/forms/fields.html](https://ejudge.lksh.ru/lang_docs/djbook.ru/rel1.9/ref/forms/fields.html).
21. Документация Django 1.9. Справочник по полям модели.  
<https://djbook.ru/rel1.9/ref/models/fields.html>.
22. Документация Django 3.0. <https://djbook.ru/rel3.0/>.
23. Документация Django. <https://docs.djangoproject.com/en/3.0/>.
24. Использование форм, не связанных с моделями.  
<https://propropregs.ru/django/ispolzovanie-form-ne-svyazannyh-s-modelyami>.
25. Как сделать бегущую строку на сайте. <https://zarabotat-na-sajte.ru/uroki-html/kak-sdelat-begushhuyu-stroku-na-sajte.html>.
26. Как стать веб-разработчиком.  
<https://eternalhost.net/blog/sozdanie-saytov/kak-stat-veb-razrabotchikom>.
27. Как отображать файлы другого формата (.docx / .pdf / .txt) в шаблонах Django.  
<https://question-it.com/questions/2740675/kak-otobrazhat-fajly-drugogo-formata-docx-pdf-txt-v-shablonah-django>.
28. Как встроить видеоплеер с тегом HTML 5 Video.  
<https://dev-gang.ru/article/htmlvideo--kak-vstroit-videopleer-s-tegom-html--video-2q3ix69qmm/>.
29. Катастрофический дефицит. Цифровому прорыву предрекли острую нехватку IT-специалистов. [https://www.dp.ru/a/2020/01/24/Katastroficheskij\\_deficit](https://www.dp.ru/a/2020/01/24/Katastroficheskij_deficit).
30. Легко настраивайте базу данных Windows SQLite и управляйте ею.  
<https://www.filehorse.com/download-sqlitestudio/>.
31. Модели. Django ORM.  
[https://ru.hexlet.io/courses/python-django-basics/lessons/models/theory\\_unit](https://ru.hexlet.io/courses/python-django-basics/lessons/models/theory_unit).
32. Обработка форм в Django. <https://texnoit.com/forms>.
33. Основы JavaScript. <https://html5book.ru/osnovy-javascript/>.
34. Отношения между моделями в Django.  
<http://sheregeda.github.io/blog/2015/01/30/models-in-django/>.
35. Постолит А. django\_world\_book на сайте [github.com](https://github.com/APostolit/django_world_book).  
[https://github.com/APostolit/django\\_world\\_book](https://github.com/APostolit/django_world_book).
36. Прохоренок, Н. А. Python 3 и PyQt 5. Разработка приложений / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2016. — 832 с.
37. Публикация сайта на PythonAnywhere.  
<https://tomekgancarczyktyko.gitbooks.io/django/content/ru/deploy/>.
38. Публикация сайта на Python и Django на ресурсе [beget.com](https://beget.com/ru/articles/webapp_python?link=webapp_python#5).  
[https://beget.com/ru/articles/webapp\\_python?link=webapp\\_python#5](https://beget.com/ru/articles/webapp_python?link=webapp_python#5).
39. Работа программистом Python: требования, вакансии и зарплаты.  
<https://pythonru.com/baza-znanij/python-vakansii>.
40. Руководство по веб-фреймворку Django. <https://metanit.com/python/django/>.

41. Руководство часть 9: Работа с формами.  
<https://developer.mozilla.org/ru/docs/Learn/Server-side/Django/Forms>.
42. Сайт для тех, кто изучает веб-технологии и создает сайты. <https://html5book.ru/>.
43. Свойство CSS background.  
<https://zarabotat-na-sajte.ru/uroki-html/kak-sdelat-fon-dlya-sajta.html>.
44. Создаем страницы для сайта на Django и публикуем на Heroku — Урок № 3.  
<https://python-scripts.com/django-simple-site-heroku>.
45. Создание блога на Django 3 для начинающих.  
<https://python-scripts.com/create-blog-django>.
46. Создание сайта на Python/Django: установка Django и создание проекта.  
<https://igorosa.com/sozdanie-sajta-na-pythondjango-ustanovka-django-i-sozdanie-proekta/>.
47. Скачать PyCharm. <https://www.jetbrains.com/pycharm/download/>.
48. Таблица цветов HTML.  
<https://zarabotat-na-sajte.ru/uroki-html/html-cveta-i-kodi-cvetov.html>.
49. Тег <style> в HTML. <https://zarabotat-na-sajte.ru/uroki-html/style-v-html.html>.
50. Уроки по фреймворку Bootstrap 4.  
<http://web-verstka.ru/uchebnyie-kursyi/uroki-po-frejmworku-bootstrap-4.html>.
51. Шаблоны Django. Наследование. <https://habr.com/ru/post/23132/>.

# ПРИЛОЖЕНИЕ

## Описание электронного архива

Электронный архив, сопровождающий книгу, выложен на сервер издательства по адресу: <https://zip.bhv.ru/9785977518079.zip>. Ссылка на этот архив доступна и со страницы книги на сайте [www.bhv.ru](http://www.bhv.ru).

Содержимое электронного архива приведено в табл. П.1.

*Таблица П.1. Содержимое электронного архива*

Папки	Описание
Глава 2	Листинги 2.1–2.9
Глава 3	Листинги 3.1–3.18
Глава 4	Листинги 4.1–4.3
Глава 5	Листинги 5.1–5.19
Глава 6	Листинги 6.1–6.50
Глава 7	Листинги 7.1–7.65
Глава 8	Листинги 8.1–8.40
Глава 9	Листинги 9.1–9.27
Глава 10	Листинги 10.1–10.30
Глава 11	Листинги 11.1–11.48
Глава 12	Листинги 12.1–12.11
Глава 13	Листинги 13.1–13.13

# Предметный указатель

## C

**Cross Site Request Forgery, CSRF** 512, 551  
**CSS-класс** 272

## D

**Django ORM** 279

## G

**GET-запрос** 529  
**GitHub** 105

## H

**HTML-документы** 40, 45  
**HTML-форма** 374, 512, 514, 515  
**HTML-шаблон** 105  
**HTTP-запрос** 105, 106

## I

**Infrastructure as a Service (IaaS)** 548  
**Integrated Development Environment, IDE** 15

## A

**Аббревиатура CRUD** 287  
**Автоматическое тестирование приложения**  
545  
**Авторизация** 509  
◊ **пользователя** 358, 474, 475

**International Standard Book Number, ISBN** 371  
**IP-адрес** 238

## O

**Object-Relational Mapping (ORM)** 279

## P

**Platform as a Service, PaaS** 548, 549  
**POST-запрос** 265, 512, 529

## S

**SQL-запрос** 279, 291, 292, 545

## U

**URL-mapper** 106  
**URL-преобразование** 422, 424, 455

**Административная панель** 407  
◊ **Django** 357, 391, 392, 402, 422, 423, 512  
◊ **сайта** 475  
**Аккаунт суперпользователя** 476  
**Архитектура**  
◊ **Model-View-Template (MVT)** 106  
◊ **Model-View-Controller (MVC)** 106

**Атака CSRF (Cross-Site Request Forgery)**

221, 276

**Атрибут**

- ◊ `exclude` 410
- ◊ `fields` 409, 410
- ◊ `fieldsets` 411
- ◊ `list_filter` 407
- ◊ `style` 173

**Атрибуты (параметры) тегов** 46

**Аутентификация** 468, 469, 476

- ◊ **пользователя** 474

## Б

**База данных SQLite** 280, 303, 314, 362

**Базовые блоки веб-приложения на Django** 106

**Базовый класс `models.Model`** 379

**Базовый шаблон** 191–194, 196, 483, 523

**Библиотека**

- ◊ **Django** 32, 34
- ◊ **SQLite** 114

**Боковая навигационная панель (главное меню)**

424

**Веб-формы** 41

**Веб-фреймворк Django** 16, 31, 32, 34, 35, 37, 43, 103, 105, 358

**Виджет** 211, 223, 225, 409, 512

- ◊ `Textarea` 255
- ◊ `TextInput` 255

◊ **Календарь** 520

◊ **по умолчанию** 223

◊ **формы** 375

**Виды стилей CSS** 172

**Внешний ключ** 405

**Внешняя таблица стилей** 176

**Внутренние стили** 175

**Восстановление доступа к аккаунту**

**пользователя** 486

**Встроенные стили** 173, 176

**Вход пользователя в систему** 500

**Выбор**

- ◊ **вложенных каталогов** 237
- ◊ **данных из списка** 228, 247
- ◊ **и загрузка файлов** 234, 239
- ◊ **файла из предложенного списка** 236, 237

**Вывод изображения** 181

## В

**Варианты отображения полей на форме** 260

**Ввод**

- ◊ **IP-адреса** 238
- ◊ **временного промежутка** 232
- ◊ **даты** 229
- ◊ **даты и времени** 230
- ◊ **десятичных чисел** 231
- ◊ **значений времени** 246
- ◊ **текста** 226
- ◊ **текста с проверкой соответствия заданным форматам** 252
- ◊ **текста типа «slug»** 246
- ◊ **текста, соответствующего регулярному выражению** 245
- ◊ **универсального указателя ресурса (URL)** 250
- ◊ **универсального уникального идентификатора UUID** 251
- ◊ **целых чисел** 240
- ◊ **чисел с плавающей точкой** 238
- ◊ **электронного адреса** 233

**Веб-сайт** 39, 40

**Веб-сервер Django** 547

**Веб-страница** 39, 47

## Г

**Гвидо Ван Россум** 15

**Гиперссылки** 56

**Главная модель** 296, 300

**Главная страница сайта** 500

**Главная таблица БД** 296, 297, 299, 383

**Главное меню сайта** 424

## Д

**Движок БД SQLite** 114, 362

**Декоратор `@register`** 404

**Динамический сайт** 40, 41

**Диспетчер URL-адресов (URLs)** 106

**Дистрибутив Python** 17

**Добавление данных в БД** 287

**Домашняя страница сайта** 484

**Дочерняя таблица** 383

◊ **в БД** 299

## З

**Зависимая модель** 296, 300

**Зависимая таблица БД** 296, 297

**Заголовок документа** 45

**Запрос**

- ◊ GET 315
- ◊ get\_queryset() 508
- ◊ HttpRequest 433
- ◊ POST 315
- Защита**
  - ◊ от межсайтовой подделки запроса 211
  - ◊ приложения от CSRF-атак 221, 276

**И**

- Идентификаторы полей** 219
- Идентификация пользователя** 483
- Изображения** 180
- Имя поля** 374
- Интегрированная среда разработки** 15, 18
- Интегрирующий объект QuerySet** 378
- Интерактивная среда разработки**
  - ◊ IDE PyCharm 21
  - ◊ программного кода PyCharm 37
- Интернет-браузер** 40, 47
- Интерпретатор Python** 16, 17, 20, 32, 33, 37
- Использование сессий** 472

**К**

- Календарь** 396
- Каскадное удаление** 297, 299
- Каскадные таблицы стилей (Cascading Style Sheets, CSS)** 39, 40, 56, 172, 176
- Класс** 58, 119
  - ◊ forms.Form 215
  - ◊ HttpResponse (ответ HTTP) 124
  - ◊ HttpResponseRedirect() 364
  - ◊ inlines 412
  - ◊ ListView 439
  - ◊ LoginRequiredMixin 505
  - ◊ Meta 376, 537
  - ◊ ModelAdmin 403
  - ◊ reverse 380
  - ◊ TemplateResponse 145, 155, 162
  - ◊ TemplateView 183
  - ◊ формы Form() 515, 533, 539
  - ◊ формы ModelForm() 533, 537–539
- Ключевое поле (id)** 375
- Ключевые поля БД** 299
- Код JavaScript** 60, 513
- Конструктор**
  - ◊ models.ForeignKey 297
  - ◊ models.ManyToManyField 302
  - ◊ models.OneToOneField() 306
- Кортеж list\_display** 404

**Л**

- Логика проверки ввода** 223

**М**

- Маркер** 49
- Маркированные списки** 49
- Маршрутизация запросов** 123, 144
- Маршруты** 124, 129
- Менеджер**
  - ◊ SQLiteStudio 35, 36, 314, 391, 400
  - ◊ баз данных SQLite 35
  - ◊ пакетов pip 16, 28, 29
  - ◊ работы с базами данных SQLiteStudio 37
- Метаданные** 376, 377
- Метка поля** 219, 220, 374
- Метод**
  - ◊ as\_view() 540
  - ◊ admin.site.register() 404
  - ◊ EmailValidator 233
  - ◊ filter() 378
  - ◊ GET 529
  - ◊ get\_absolute\_url() 386
  - ◊ is\_valid() 264
  - ◊ objects.all() 378
  - ◊ POST 529
  - ◊ RegexValidator 245
  - ◊ save() 377
  - ◊ URLValidator 250
  - ◊ validate\_slug 246
  - ◊ validate\_unicode\_slug 246
  - ◊ ValidationError 245
- Механизм**
  - ◊ валидации 262
  - ◊ переадресации 140
- Миграция** 281, 282, 298, 306, 308, 360, 474
  - ◊ данных 389, 407
- Множественный выбор данных из списка** 243, 249
- Модели** 107, 366, 371, 394
  - ◊ авторизации 474
  - ◊ данных 278, 297, 360, 373, 385, 387, 391, 402, 406, 422, 423, 537
- Модель** 279
  - ◊ данных Django 280
- Модуль** 119

**Н**

- Настройка
  - ◊ DEBUG 551
  - ◊ SECRET\_KEY 551
  - ◊ шаблонов 145
- Несвязанная форма 515
- Нумерованные списки 50

**О**

- Облачные решения 548
- Облачный сервис Heroku 548
- Обновление объекта в БД 290
- Обобщенные классы 512, 537
  - ◊ редактирования форм 545
  - ◊ базовые классы визуализации данных 438
  - ◊ создания страниц 537
  - ◊ отображения списков 448
- Общие аргументы полей моделей 374
- Объект 371
  - ◊ HttpResponseRedirect 124, 277
  - ◊ QuerySet 288, 289, 378
  - ◊ request 123, 124
  - ◊ UploadedFile 234, 239
  - ◊ запроса пользователя 119
  - ◊ формы 216
- Объектно-ориентированный язык JavaScript 58
- Объекты 279
  - ◊ Python 107
  - ◊ модели данных 308
- Объявление стиля 172
- Окно
  - ◊ администрирования сайта 394
  - ◊ терминала PyCharm 107
- Окружение развертывания 548
- Основные способы доступа к данным модели 370
- Основные типы полей таблиц 370
- Ответ (response) 123
- Отключение атрибута required 263
- Отношение
  - ◊ «многие ко многим» 375
  - ◊ «один ко многим» 375, 379
- Отношения между моделями 371
- Очистка данных 515

**П**

- Пакетный менеджер pip 18
- Панель
  - ◊ администратора сайта 393, 476, 484
  - ◊ администрирования Django 411
  - ◊ администрирования сайта 394, 400
  - ◊ календаря 415
- Параметр 159
  - ◊ DATABASES 280
  - ◊ ENGINE 280
  - ◊ label 220
  - ◊ NAME 280
  - ◊ page\_obj 451
  - ◊ request 470
  - ◊ success\_url 539
  - ◊ widget 255
- Параметры
  - ◊ корректности вводимых данных 262
  - ◊ строки запроса 138
  - ◊ передаваемые через интернет-адрес 137
  - ◊ передаваемые через строку запроса 137
- Пароль
  - ◊ пользователей 492
  - ◊ суперпользователя 505
- Первичный ключ 302, 305, 386, 388, 541
  - ◊ в таблице базы данных 283
  - ◊ для модели 375
- Переадресация 140, 142
  - ◊ временная 140
- Передача в шаблон сложных данных 161
- Переменная
  - ◊ context 473
  - ◊ session 470
  - ◊ urlpatterns (шаблон URL) 125
- Платформа как Сервис (PaaS) 548
- Подсказки 260
- Поиск хостинга 547, 583
- Поле
  - ◊ exclude 538
  - ◊ fields 538
- Пользовательские аккаунты 468
- Поля формы 215, 223, 538
- Порядок следования полей на форме 258
- Постоянная переадресация 141
- Постраничный вывод (Pagination) 448
  - ◊ длинных списков 468
- Постраничный показ данных 438
- Представление (view) 106, 107, 182, 308, 402, 425, 427, 468, 472, 505, 514, 539

Представления (views) 123, 144, 158, 361, 365, 366, 423, 481  
 Приложение  
     ◊ длинных списков 468  
     ◊ «Администратор сайта» 365, 366  
     ◊ Django 360  
     ◊ Django Admin 391, 503  
     ◊ PyCharm 358–360  
     ◊ SQLiteStudio 284, 320  
 Присвоение стилей полям формы 272, 274  
 Пробные бесплатные тарифы 549  
 Проверка  
     ◊ введенных значений на стороне клиента 264  
     ◊ данных 515  
     ◊ корректности данных на стороне сервера 264  
 Программа SQLiteStudio 283  
 Программная оболочка PyCharm 147, 176  
 Проект  
     ◊ Django 107, 109  
     ◊ Open Source 105  
     ◊ Python 26  
 Протокол HTTP 470  
 Публичный веб-сервер 547

**P**

Разграничение уровней доступа 468, 469  
 Регистрация приложения в проекте Django 361  
 Регулярное выражение 127–129, 132, 236, 245, 439, 443, 453, 456, 468  
 Редактирование объектов модели 315, 319  
 Реляционная база данных SQLite 114  
 Репозиторий Python Package Index (PyPI) 28  
 Родительская таблица 383  
     ◊ в БД 299, 300

**C**

Свойство  
     ◊ help\_text 259  
     ◊ initial 257  
     ◊ формы error\_css\_class 272  
     ◊ формы required\_css\_class 272  
 Связанная таблица 383  
 Связанная форма 515  
 Связанные данные 299  
 Связующая таблица 303, 384  
 Связь  
     ◊ «многие ко многим» 301–303, 305, 384, 397  
     ◊ «один к одному» 305, 306  
     ◊ «один ко многим» 296, 305, 382, 383, 387, 397, 405, 503

Селектор 172  
 Серверная часть системы 514  
 Сертификат SSL 549  
 Сессии 468, 469, 470  
 Система  
     ◊ аутентификации и авторизации Django 474  
     ◊ сброса пароля 486  
     ◊ смены пароля 489  
     ◊ управления базами данных (СУБД) 279  
 Скриптовый язык JavaScript 39, 41  
 Скрипты  
     ◊ Django 107  
     ◊ JavaScript 172, 176  
 Смена пароля пользователя 489, 490, 492  
 Содержимое таблицы БД 314  
 Соединительная таблица 302  
 Сопоставление URL-адресов 439  
 Специальные теги 203  
 Списки определений 51  
 Среда разработки PyCharm 16, 32  
 Статические файлы 172, 180  
 Статический сайт 40  
 Статусные коды 142  
 Страница  
     ◊ администратора сайта 365  
     ◊ аутентификации 481  
     ◊ выхода из системы 485  
     ◊ идентификации пользователя 500, 501  
     ◊ профиля пользователя 483  
 Структура проекта Django 107  
 Суперпользователь 391–394, 422, 475  
 Схема Model-View-Controller (MVC) 104  
 Сценарии JavaScript 58, 59  
 Счетчик посещений домашней страницы 469

**T**

Таблица базы данных 281, 308, 356, 373  
 Таблицы-«источники» 302  
 Тег label 513  
 Теги 39, 45, 47, 48, 50, 51, 59  
 Текст «slug» 246  
 Тело документа 45  
 Терминал PyCharm 281, 283, 297, 299, 302, 306, 359, 360, 389, 392, 491, 504  
 Тестирование приложения 361  
 Тип запроса  
     ◊ GET 277  
     ◊ POST 277  
 Типовые модульные сетки 53  
 Типы отношений в БД 296

**Типы полей**

- ◊ в моделях данных Django 284
- ◊ в различных СУБД 286, 287
- ◊ в формах Django 222

**У****Удаление**

- ◊ данных из БД 291
- ◊ объектов модели 321

**Универсальный указатель ресурса (URL) 250****Универсальный уникальный идентификатор  
UUID 251****Учетная запись superuser (суперпользователь)  
392****Ф****Файл миграции 282, 389, 504****Файлы cookie 104, 470, 471****Форма 211, 483, 524, 531**

- ◊ HTML 512

**Формы 209, 220, 278, 358, 391**

- ◊ аутентификации 474

**Фреймворк SQLite Studio 16****Функции-представления 130****Функция 119**

- ◊ «обратного просмотра» (reverse lookup) 446
- ◊ path 128
- ◊ path() 127, 134, 135, 137
- ◊ re\_path 135
- ◊ re\_path() 127, 128
- ◊ render (предоставлять) 150
- ◊ render() 145, 155, 433, 439
- ◊ reverse\_lazy() 539
- ◊ url() 439, 443, 453, 456

**Х****Хешированный пароль 104****Хеш-функция 104****Хостинг-провайдер 548, 549****Ч****Чтение данных из БД 288****Ш****Шаблон**

- ◊ HTML-страницы 539
  - ◊ регулярного выражения 236
  - ◊ страницы 310
- Шаблоны 107, 145**
- ◊ URL-адресов 540
  - ◊ преобразования URL-адресов 468

**Э****Электронный адрес 233****Элемент input типа type="submit" 513****Я****Язык**

- ◊ HTML 40, 45, 512, 513
- ◊ Perl 41, 42
- ◊ Python 15–17, 19, 21, 29, 35, 37, 42, 43, 103
- ◊ SQL 279, 287
- ◊ сценариев общего назначения PHP 41, 42



ИНТЕРНЕТ-МАГАЗИН  
**BHV.RU**  
КНИГИ, РОБОТЫ,  
ЭЛЕКТРОНИКА

**Интернет-магазин издательства «БХВ»**

- Более 25 лет на российском рынке
- Книги и наборы по электронике и робототехнике по издательским ценам
- Электронные архивы книг и компакт-дисков
- Ответы на вопросы читателей



**Скидка 15%**

на бумажные книги  
по промокоду: **JANE22**

# Python, Django и Bootstrap

## для Начинающих



**Постолит Анатолий Владимирович** — доктор технических наук, профессор, академик Российской академии транспорта, лауреат Всероссийского конкурса «Инженер года». Профессиональный программист, автор книг компьютерной тематики, в том числе «Основы искусственного интеллекта в примерах на Python», «Python, Django и PyCharm для начинающих» и более 100 научных публикаций. Преподавал в Московском государственном автомобильно-дорожном техническом университете (МАДИ). Занимался разработкой и внедрением информационных систем для транспортного комплекса Москвы и Московской области, для транспортного обслуживания зимних Олимпийских игр в г. Сочи, систем оплаты проезда и информирования пассажиров городского общественного транспорта. Специализируется на создании информационных систем на основе MS SQL Server, MS Visual Studio, Bluetooth-технологий, а также инновационных проектов с использованием Python, Django, Bootstrap, PyCharm и различных специализированных библиотек.

## Осваиваем современный и удобный инструментарий для создания веб-приложений

Книга посвящена вопросам разработки веб-приложений с использованием языка Python, фреймворков Django, Bootstrap и интерактивной среды разработки PyCharm. Рассмотрены основные технологии и рабочие инструменты создания веб-приложений. На простых примерах показана обработка и маршрутизация запросов пользователей, формирование ответных веб-страниц. Рассмотрено создание шаблонов веб-страниц и форм для пользователей. Показано взаимодействие пользователей с различными типами баз данных через модели. Описана работа с базами данных через встроенные в Django классы без использования SQL-запросов. Приведен пошаговый пример создания сайта — от его проектирования до формирования программных модулей и развертывания сайта в Интернете с базами данных SQLite и MySQL.

- Веб-технологии
- Инstrumentальные средства для разработки веб-приложений
- Знакомство с фреймворком Django
- Знакомство с фреймворком Bootstrap
- Интерактивная среда разработки PyCharm
- Обработка и маршрутизация запросов
- Шаблоны веб-страниц
- Формы и модели данных
- Веб-сайт и веб-интерфейс для пользователей
- Встроенная панель для администрирования сайта
- Пользовательские формы
- Публикация сайта в Интернете



Примеры из книги можно скачать по ссылке <https://zip.bhv.ru/9785977518079.zip>, а также на странице книги на сайте <https://bhv.ru>.

ISBN 978-5-9775-1807-9

9 785977 518079

bhv®

191036, Санкт-Петербург,  
Гончарная ул., 20  
Тел.: (812) 717-10-50,  
339-54-17, 339-54-28  
E-mail: mail@bhv.ru  
Internet: www.bhv.ru